Bayes Linear Strategies for the Approximation of Complex Numerical Calculations Arising in Sequential Design and Physical Modelling Problems.

JONES, MATTHEW,JAMES

Bayes Linear Strategies for the Approximation of Complex Numerical Calculations Arising in Sequential Design and Physical Modelling Problems.

Matthew Jones

A Thesis presented for the degree of
Doctor of Philosophy

Statistics and Probability Group
Department of Mathematical Sciences
University of Durham
England

October 2017

Bayes Linear Strategies for the Approximation of Complex Numerical Calculations Arising in Sequential Design and Physical Modelling Problems.

Matthew Jones

Submitted for the degree of Doctor of Philosophy
October 2017

Abstract

In a range of different scientific fields, deterministic calculations for which there is no analytic solution must be approximated numerically. The use of numerical approximations is necessary, but introduces a discrepancy between the true solution and the numerical solution that is generated. Bayesian methods are used to account for uncertainties introduced through numerical approximation in a variety of situations. To solve problems in Bayesian sequential experimental design, a sequence of complex integration and optimisation steps must be performed; for most problems, these calculations have no closed-form solution. An approximating framework is developed which tracks numerical uncertainty about the result of each calculation through each step of the design procedure. This framework is illustrated through application to a simple linear model, and to a more complex problem in atmospheric dispersion modelling. The approximating framework is also adapted to allow for the situation where beliefs about a model may change at certain points in the future.

Where ordinary or partial differential equation (ODE or PDE) systems are used to represent a real-world system, it is rare that these can be solved directly. A wide variety of different approximation strategies have been developed for such problems; the approximate solution that is generated will differ from the true solution in some unknown way. A Bayesian framework which accounts for the uncertainty induced through numerical approximation is developed, and Bayes linear graphical analysis

is used to efficiently update beliefs about model components using observations on the real system. In the ODE case, the framework is illustrated through application to a Lagrangian mechanical model for the interaction between a set of ringing bells and the tower in which they are hung; in the PDE case, the framework is illustrated through application to the heat equation in one spatial dimension.

Declaration

The work in this thesis is based on research carried out at the Statistics and Probability Group, the Department of Mathematical Sciences, Durham University, UK. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Acknowledgements

I would like to thank the Engineering and Physical Sciences Research Council and Shell Global Solutions for providing me with the opportunity to write this thesis through a CASE studentship. I would also like to thank the Durham CDT in Energy for providing additional support for my research. Thanks also, of course, to my supervisory team: to Michael Goldstein for giving guidance and a general Bayesian education; to Phil Jonathan for his advice and support, and for his almost unbounded enthusiasm; and to David Randell for his invaluable assistance with the practicalities of everything.

Thanks to Phil and David, and to everybody in the Statistics and Chemometrics group at Shell for giving me the opportunity to work with them as an intern, and thanks to the group's internal and external collaborators for letting me loose on their projects. In particular, thanks to Bill Hirst for the many interesting discussions about LightTouch and the dynamics of boomerang throwing, and to Rakesh Paleja for his mentorship and for his accurate summaries of the world around us.

Last, but definitely not least, thanks to all of my friends and family for their support throughout the years. Thanks to Lara for her patience and organisational assistance, and to my parents for the same, and for allowing me to stay with them so often.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In a very wide range of scientific and industrial problems, the main aim is the analysis of the behaviour of a complex system using a quantitative model; scientists specify a model as a description of the system, tune the parameters of the model so that it gives predictions which match the behaviour of the system, and then use the model's predictions to make decisions concerning future system behaviour. Both systems and models vary greatly in complexity between fields of study.

Generally, many different sources of uncertainty will exist within a given problem. Models typically give a simplified representation of real-world behaviour, leaving uncertainty about the relationship between the model and the real world. Even if the model is believed to be a perfect description of the system under study, there is generally still uncertainty about which model input settings give the best description of the real data.

In this thesis, we consider the specification and analysis of models which take account of all of our uncertainties about a given system, and develop methodology for the design of experiments on the system which take into account these uncertainties and decisions about the system which we will make using the model and the observed data. In this introductory section, we outline a number of problems encountered in the energy industry which contain a number of different sources of uncertainty and thus would be suitable for such an analysis.

In Section 1.1, we consider an atmospheric dispersion problem, in which sources of gas must be mapped from remotely-observed concentration data, and decisions must

be made about further exploration or mitigation of these sources. In Section 1.2, we outline the problem of assessing the level of seismic activity in a given region, in which a model for the rates and magnitudes of seismic events must be developed, before being used to inform decisions about infrastructure developments, such as strengthening buildings. Finally, in Section 1.3, we consider a problem in which extreme ocean characteristics must be modelled in order to make decisions about the design of offshore assets. An outline of the structure of the remainder of the thesis is given in Section 1.4.

1.1 Atmospheric dispersion modelling

The main problem which we will consider in this thesis is one in the field of atmospheric dispersion modelling; in such a problem, we seek to use observations of the concentration of a given gas species made at particular, known points to learn about the locations and emission rates of any sources of this gas within a given region of interest. This problem is of interest for scientists working in a number of different areas:

- in the oil and gas industry, naturally-occurring sources of methane can indicate the presence of underground hydrocarbon reserves, and so mapping these sources is a useful exploration tool [Hirst et al., 2013]. Airbourne measurements of the atmospheric concentration of methane can be obtained quickly and relatively inexpensively, and can be used to target the use of more costly exploration techniques, such as seismic surveys, at regions which are more likely to contain hydrocarbons;

- often, industrial processes must comply with environmental regulations which stipulate allowable emission levels for particular gas species; it is useful for any monitoring procedure to be able to determine the origin of emissions which breach the allowed limits, as this allows the monitoring body to determine responsibility for the emissions (if necessary) and also allows for targeted remedial action [Hirst et al., 2017];

- defence agencies in many countries believe that attacks on their soil may involve the release of airbourne toxins in densely populated areas; they are therefore interested in being able to rapidly identify the source of the release so that risk to life can be minimised, and so that steps can be taken to isolate the sources of the release [Senocak et al., 2008].

The motion of gas particles within the atmosphere is an extremely complex process, as can be determined by considering even something as simple as the motion of leaves being blown by the wind. Given that there will always be a limit to our ability to specify or to measure the process and the parameters which drive it (since the concentration and, for example, the wind field are infinite dimensional functions over any continuous domain), we can see before we have even specified a model that we will always remain uncertain about some aspects of the system.
We wish to develop a model for the concentration of gas at a range of locations as a function of a set of parameters which we might reasonably be able to specify (preferably, though not necessarily, ones that we can specify from measurements made on the system); we then wish to use this model, in conjunction with a specification for all of our uncertainties about its relationship to the system, to draw conclusions about likely source locations, before using this information to make decisions. In the following sections, we will review the existing literature on atmospheric dispersion modelling, and present some standard models for this situation.

1.1.1 General model: The advection-diffusion model

The standard mathematical model for the behaviour of gas concentrations under given atmospheric conditions is the advection-diffusion model (also referred to in the literature as the convection-diffusion model); Stockie [2011] gives an introduction to the field of atmospheric dispersion modelling. This model is derived from the law of conservation of mass, which can be written as a differential form for the mass concentration $u(x,t)$ (in kg/m^3) at a location $x = (x, y, z) \in \mathbb{R}^3$ (metres) and time t (seconds) as follows

$$\frac{\partial u}{\partial t} + \nabla . J\left(u\left(x,t\right)\right) = h\left(x,t\right) \tag{1.1.1}$$

where $J(u)$ (kg/m^{2}s) is the mass flux of gas at a particular location and concentration level, and $h(x,t)$ (kg/m^{3}s) is a source or sink term. The conservation law states that the rate of change of the mass concentration (hereafter abbreviated to concentration) within a given infinitesimal volume is controlled by the mass flux over the boundary of the volume, and any sources or sinks within the volume. Integrating equation (1.1.1) over a volume V and applying Stokes' theorem gives the following, more intuitive conservation law for a general volume

$$\frac{\partial}{\partial t}\left(\int_V u(x,t)\,dV\right) = -\int_{\partial V} J(u(x,t)).n(x)\,d\sigma(x) + \int_V h(x,t)\,dV$$

which states that the rate of change of the mass of the gas species within volume V is equivalent to the negative flux of gas across its boundary ∂V (where $n(x)$ is an outward normal vector at x) plus the total contribution from any sources or sinks within V.

The model is completed by specifying a form for the mass flux $J(.)$; we assume that this is due to the combined effects of an advection process (transport due to the wind) and a diffusion process (transport due to turbulent eddy motion in the atmosphere). Fick's law [Fick, 1855] states that the diffusive flux is proportional to the concentration gradient, and we assume a simple linear advective flux, allowing us to write $J(u(x,t)) = -K(x)\nabla u + w(x)u$, where $K(x)$ is a matrix of diffusion constants and $w(x)$ is a transport vector, and resulting in the following partial differential equation (PDE)

$$\frac{\partial u}{\partial t} + \nabla.(w(x)u) - \nabla.(K(x)\nabla u) = h(x,t)\ . \tag{1.1.2}$$

This is the advection-diffusion equation (ADE); when supplemented with an appropriate set of initial and boundary conditions, it can be shown that this is a well-posed problem with a unique solution.

Sources of uncertainty If our model for the dispersion of gas is given by the solution to (1.1.2), and we have access to a finite number of observations of the concentration and wind fields at known locations, then before we can infer properties of $h(x,t)$, we must specify our beliefs about each of the following:

- the wind field at all locations and times;

- the diffusion matrix (possibly as a function of location and wind field) at all locations and times;
- the likely properties of the source field (eg: constrained to be on the ground, locations that are ruled out);
- the behaviour of processes which the model does not adequately capture (eg: small-scale turbulence).

All of these aspects are likely to be subject to significant uncertainty.
An additional and important source of uncertainty in the case of this model arises from the fact that we cannot solve the differential equation (1.1.2) for realistic cases, and so must approximate the solution in some way. There are many ways in which this can be done; we consider two such ways in the following subsections. In Section 1.1.2, we consider how the equation can be solved using a finite-dimensional approximation, and in Section 1.1.3, we consider an exact solution which can be obtained using a particular (unrealistically simple) set of initial and boundary conditions.

1.1.2 Simplified model: Numerical solution

While differential equation models are commonly postulated as suitable models for physical (and other; economic etc.) systems, it is rare that we may derive an exact solution to such equations; it is common, therefore to look for finite-dimensional approximations to the infinite-dimensional solutions of these equations which converge in the limit to the true solution of the original equations. Such finite-dimensional approximations are referred to as numerical schemes, or numerical solvers.
A large variety of such schemes have been developed over time; for ordinary differential equations (ODE), which feature only derivatives with respect to a single independent variable, Euler and Runge-Kutta schemes are perhaps the most widely used [Hairer et al., 1993], whereas in the case of PDE, the most commonly used are the finite-element (see for example Iserles [2008]) and finite-volume schemes (see, for example, Eymard et al. [2000]).
The finite-volume approximation of the ADE is a popular choice, in part because this scheme has a direct relationship with the conservation law from which the equation

is derived. To use this approximation, we impose a polygonal (for example, cubic, tetrahedral) mesh on the domain and derive discrete approximations to the coefficient functions in (1.1.2) on the mesh: these discrete coefficient values are then used to derive a system of equations relating a set of discrete, unknown concentration values to the discretized boundary, source and parameter functions.

Additional sources of uncertainty Choosing to use such a numerical scheme as a model for an atmospheric dispersion process introduces additional uncertainty, owing to the fact that the numerical predictions provided by our discrete model are only an approximation to the true, underlying solution of (1.1.2). This additional uncertainty arises through:

- the use of an averaged value to summarize the originally infinite-dimensional behaviour of the coefficient functions over a given volume and time element of the input domain;
- the use of an averaged value to summarize the concentration over the volumes used to discretize the input domain.

This additional uncertainty will manifest itself as an additional discrepancy between our numerical predictions for the gas concentrations, and the real behaviour of the gas; in addition to the difference between the initial model (the real solution to (1.1.2)) and the real world due to effects that this model does not capture, we now have an additional discrepancy between the numerical model which we actually use, and the (unknown) underlying model that we would like to have used.

1.1.3 Simplified model: The Gaussian plume

A popular alternative strategy in the field of atmospheric dispersion modelling is to make use of a solution to (1.1.2) which is available under a much more restrictive class of initial and boundary conditions. This solution is known as the Gaussian plume and its simple (albeit non-linear, for some important parameters) form has made it a very attractive model for a variety of authors: Pasquill [1971], Draxler [1976], Hirst et al. [2013] and Kennedy and O'Hagan [2001] all use this as a model

for the transport of particulate matter.

Stockie [2011] describes the necessary additional assumptions and derives the plume: for a collection of sources of a particular gas species at locations $\{g_1, \ldots, g_{n_\psi}\}$ (with $g_j = (g_{jx}, g_{jy}, g_{jz})^{\mathrm{T}}$ the three-dimensional location of each source) with scalar emission rates $\{\psi_1, \ldots, \psi_{n_\psi}\}$, and constant wind velocity field $w = (w_x, w_y)^{\mathrm{T}}$, the concentration at spatial location $x = (x, y, z)^{\mathrm{T}}$ is modelled as

$$u\left(x, \{g_j, \psi_j\}, w\right) = \sum_{j=1}^{n_\psi} a\left(\omega(x, g_j, w), g_j, w\right) \psi_j + b\left(x, w\right)$$

where $\omega(x, g, w)$ is the vector from the source to the measurement location projected onto the wind direction and normalised by the wind magnitude

$$\begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \frac{1}{|w|} \begin{pmatrix} w_x & w_y & 0 \\ -w_y & w_x & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_x - g_x \\ x_y - g_y \\ z \end{pmatrix} .$$

$b\left(x, w\right)$ is the background concentration of the gas species at x under wind conditions w (that is, the component of the concentration measurement not attributable to the sources at g), and the individual coupling constants are computed as

$$\begin{aligned} a\left(\omega, g, w\right) = \; & \frac{1}{4\pi|w|\sqrt{\kappa_y \kappa_z}} \exp\left(-\frac{\omega_y^2}{4\kappa_y}\right) \\ & \times \left[\exp\left(-\frac{(\omega_z - g_z)^2}{4\kappa_z}\right) + \exp\left(-\frac{(\omega_z + g_z)^2}{4\kappa_z}\right)\right] \\ & \times \left[\exp\left(-\frac{(2D - \omega_z - g_z)^2}{4\kappa_z}\right) + \exp\left(-\frac{(2D - \omega_z + g_z)^2}{4\kappa_z}\right)\right] \end{aligned}$$

where D is the height of the atmospheric boundary layer (effectively the 'top of the atmosphere', for the purposes of gas transport), and κ_y and κ_z are related to the diffusion constants in equation (1.1.2) as follows

$$\kappa_y = \frac{1}{|w|} \int_0^{\omega_x} K_y\left(\eta\right) d\eta \; .$$

In practice, κ_y and κ_z are usually specified directly as the plume 'variances'; a popular choice is

$$\kappa_y = \left(\omega_x \tan\left(\gamma_h\right) + h\right)^2 \qquad \kappa_z = \left(\omega_x \tan\left(\gamma_v\right)\right)^2$$

where γ_h and γ_v are constants which are determined from atmospheric data, and h is the half-width of the source.

Additional sources of uncertainty While the Gaussian plume is a popular modelling choice because of its simplicity and low computational burden, its use also introduces additional discrepancies between the model and the real world which should be accounted for when using such a model to make predictions, inferences or decisions:

- to obtain the simplified solution, it is necessary to reduce the representation of the wind and diffusion parameters down to the scalar values w and κ for each direction. This constitutes a huge simplification of our description, and introduces even greater systematic discrepancies between the model and the real world than we would see if using a numerical solution of the form described in Section 1.1.2;

- in deriving the solution, it is necessary to assume stationarity; this means that the Gaussian plume describes instantaneous transport of matter from the source to the measurement location. This is clearly not a valid assumption for individual particles of gas, and so in practice, this restricts the use of the Gaussian plume model to cases where the gas has been transported under steady atmospheric conditions for a long period of time.

In practice, the steady-state assumption used in deriving the Gaussian plume model restricts its applicability somewhat. In problems where airbourne concentration measurements are made over a short period of time (a few hours at most) in a stable atmosphere, it generally gives a reasonably good description of the data, since the average behaviour of the gas over long length-scales (tens of kilometres) resembles a plume. However, in problems where measurements are made at short range (may be a few hundred metres), the behaviour of the wind is much more variable, and the direction of transport may change drastically over the course of a few minutes; unless the local atmosphere is exceptionally calm, the Gaussian plume is generally not a good model in such situations.

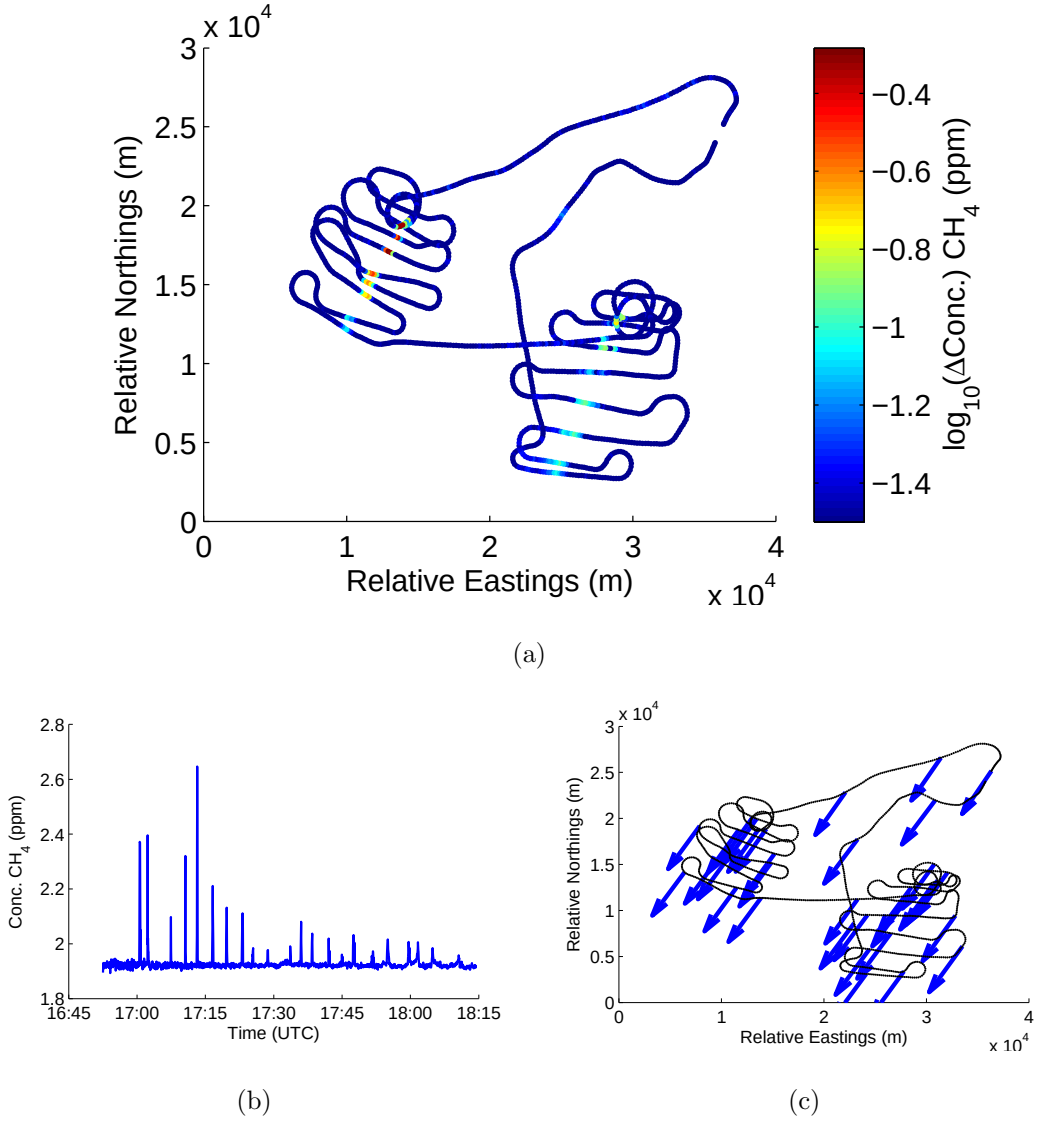


Figure 1.1: Data obtained from an aircraft-mounted concentration sensor flown downwind of two landfill sites in Canada: Figure 1.1(a) shows $\log(u - \min(u))$ plotted onto the flightpath of the aircraft; on this log scale we can see the shape of the plume originating at each of the two landfill locations. Figure 1.1(b) shows the observed concentration as a function of time, with clear peaks every time the aircraft intersects one of the plumes. Figure 1.1(c) shows the wind vector (from UKMO data) for every 50^{th} measurement location; note that this data has been highly regularised, and so is likely to miss local deviations from the prevailing direction.

1.1.4 Example data

The data presented in Hirst et al. [2013] provide an excellent illustration of gas transport under relatively static wind conditions; over the course of about 2 hours, an aircraft fitted with a concentration sensor is flown downwind of two landfill sites, which are known to be potent sources of anthropogenic methane (CH_4). The concentration is measured every 3 seconds, and the trace plot of the observed concentrations (in parts per million) in time is shown in Figure 1.1(b). The atmospheric background concentration of methane is about 1.9ppm, and there is clearly measurement noise on the data; however, it is clear that the large peaks in the data correspond to anomalous, non-natural contributions. The log of the increase in concentration over the minimum observed value is plotted on the flight path of the aircraft in Figure 1.1(a), to enhance our ability to see the shape of the plumes emanating from the landfills. The arrows in Figure 1.1(c) show the approximate wind direction for each observation, obtained from model data supplied by the UK MET Office; it is clear from this data that the prevailing wind direction is relatively static for the duration of the flight.

We see in Figure 1.1(a) that under relatively static wind conditions, the behaviour of the gas in the real system looks much like that of the stationary Gaussian plume model outlined in Section 1.1.3. However, we can also see some of the types of discrepancy discussed in the previous sections manifesting themselves:

- Downwind of the Eastern landfill, we see that the centre of the observed plume appears to bend between flight lines as we move away from the source; local trends in the wind between flight lines result in systematic deviations from the standard model. The wind information that we have is itself the product of smoothing, and so even if our model allowed for this effect, it is unlikely that we would have wind information of a high-enough resolution to be able to explain it.

- We can also see evidence of turbulent effects causing systematic deviations within individual flight lines; for example, again, downwind of the Eastern landfill. Since the Gaussian plume is a stationary solution of the governing

equation, we expect long-term average behaviour to be plume-like, but for an individual snapshot, we expect systematic variability like this over short length-scales.

1.2 Seismic hazard risk assessment

In recent years, there has been a marked increase in the number of seismic events which have been measured in the area around the Groningen gas field in the Netherlands; van Thienen-Visser and Breunese [2015] give an overview of the history of seismic activity and production for this field. The largest measured event over the past 20 years had a magnitude of 3.6, and several recent events have been large enough to cause damage to buildings in the area above the field, and to present a risk of harm to local residents. Figures 1.2(a) and 1.2(b) show event data recorded in the field up to mid-March 2014.

The company responsible for the operation of the gas field is keen to explore the relationship between the extraction of gas from the field and the observed events, in order to determine the extent to which the increases in rate and magnitude are being driven by gas extraction. Any relationships which are found between properties of the field and rate and magnitude of events will then be used to predict the properties of future events, before these predictions are fed into a further model which describes the behaviour of local structures during seismic events. The output from this analysis will then be used to assess the risk posed to property and to life from continued extraction.

1.2.1 Modelling

The relationship between local human activity and the rate and magnitude of seismic events is relatively poorly understood (certainly in comparison to the relationship between gas sources and concentration fields- see Section 1.1); the lack of a generally accepted model within the community of domain experts presents additional challenges when accounting for all of the uncertainty about rate and magnitude. Modelling this problem will involve:

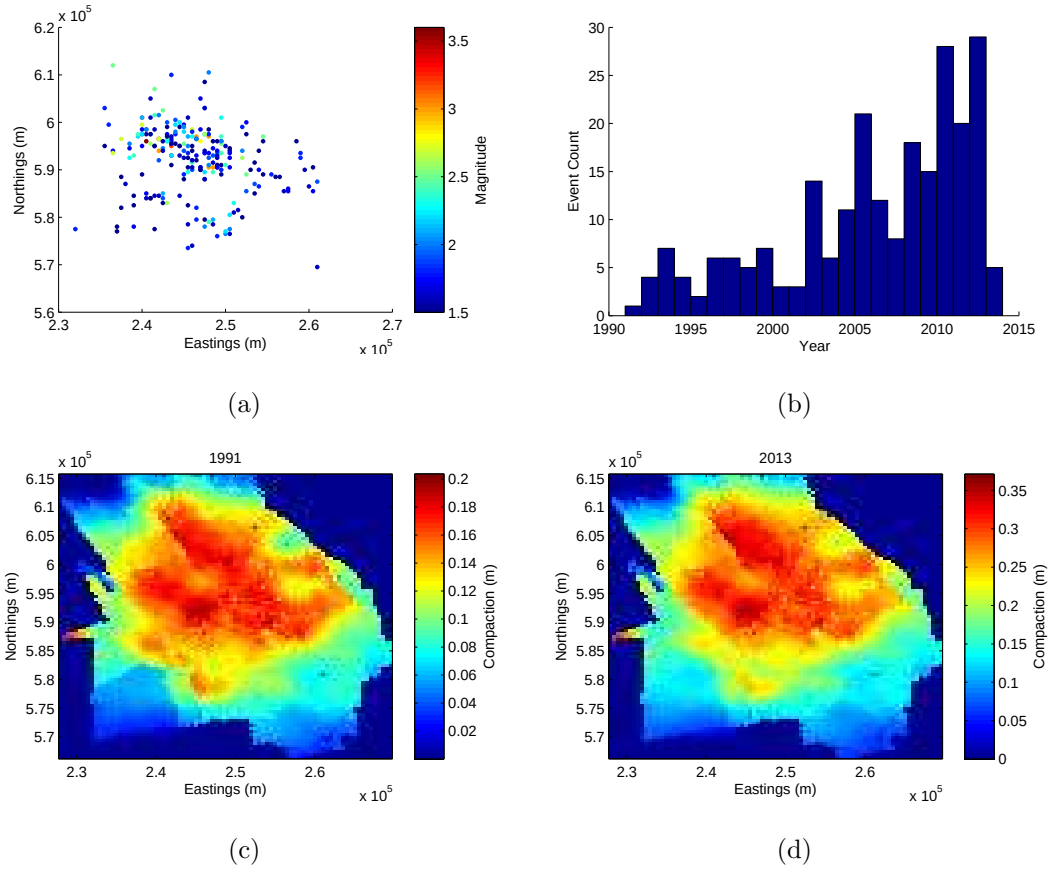


Figure 1.2: Seismic events recorded around the Groningen gas field up to mid-March 2014: Figure 1.2(a) marks the estimated location of each of the measured events, and the colour scale indicates the measured magnitude; Figure 1.2(b) shows a histogram of the event count for each year from 1991 to 2014, showing a steadily increasing rate of occurrence; Figures 1.2(c) and 1.2(d) show the estimated reservoir compaction levels in the reservoir in the first and final years of the measured event catalogue, demonstrating a large increase.

- specifying a set of candidate models for rate and magnitude as a function of field properties;
- using historical event data to learn about uncertain parameters in these models;
- using historical event data to compare different models against each other, identifying which field properties are useful in modelling the events, and which are not;
- using predictions from the best available models in conjunction with models for the damage caused by earthquakes to quantify the risks of different production strategies.

One field property which is of particular interest is compaction. The extraction of gas from the reservoir has, over decades, caused compaction in the reservoir to occur. While this compaction cannot be directly measured, it can be estimated from subsidence measurements made in areas above the reservoir, and there is great interest in using these estimates of reservoir-level compaction as covariates to predict event properties. Bourne and Oates [2015] use a spatial point-process model, in which compaction is used as a linear predictor for the rate of occurrence, with some success. Bourne et al. [2015] consider the risk quantification problem in detail.

Sources of uncertainty In this problem, we will encounter some of the same sources of uncertainty as outlined in Section 1.1; our model will be an imperfect description of the behaviour of the system (even more so than in the atmospheric dispersion problem), and so we must make sure that we account for possible differences between model predictions and observed data. In this context, we may also encounter the following sources of uncertainty:

- many of the predictors that we wish to use to model the seismic activity are themselves subject to uncertainty, due to the fact that we cannot measure them directly. This uncertainty must be handled both when fitting and predicting from the model;

- since there is disagreement between experts as to the form that the model should take, we must also be able to handle uncertainty about which model should be used to reproduce system behaviour. It may be the case that there is a single 'best' model which we use to represent the system, but alternatively, it may be that different models give varying qualities of prediction in different parts of their input spaces.

1.3 Extreme ocean wave modelling

Another area of interest in the oil and gas industry is that of offshore platform design. Offshore platforms are used to extract hydrocarbons from reservoirs located below the sea: they must often be operated in extremely challenging ocean environments, and so it is crucial that they are designed to withstand the worst conditions that they could experience in a given area.

This need to design to withstand harsh environments must be offset against the cost of doing so: companies must be run so that they are profitable, and so oil will only be extracted from reservoirs where it is economical to do so. One of the main factors in determining the economic viability of extraction from a given reservoir is the cost of the necessary equipment: therefore, reducing the cost of designing safe platforms increases the profit that can be made through operating offshore fields, and opens up the possibility of exploration in areas which were previously deemed economically inviable.

Much work has been done on modelling ocean waves in the past, and many companies employ specialists who develop models that use atmospheric and ocean conditions to simulate the characteristics of waves. These ocean simulators are tuned by running them as hindcasts, meaning that they are run using historic atmospheric data as inputs in order to re-create conditions that were actually observed; if reliable forecasts for future atmospheric behaviour are available, then a simulator can be run to predict the worst waves that may occur in the future, and offshore structures can be designed against these. In situations where suitable models or atmospheric forecasts are not available, then designs are chosen by multiplying the

worst historically-observed waves by industry-standard safety factors. These safety factors are typically conservative, and so while the resulting designs are safe, they are wasteful in many situations.

There has been much recent interest in developing models which can handle uncertainties about future waves when generating designs, and can represent the effects of storm parameters on the worst observed waves; in the following section, we consider the way in which this modelling is usually carried out, and the sources of uncertainty that need to be considered.

1.3.1 Modelling

Ocean behaviour is recorded in terms of sea states; these are periods of time (typically 1 or 3 hours) in which ocean characteristics (e.g. wave height, wave period, power spectrum) are believed to be roughly stationary. Within these sea states, ocean behaviour is recorded in terms of a number of summary statistics, for example:

- the significant wave height H_S is defined as 4 times the standard deviation of the ocean surface elevation over a given sea state;
- the peak wave period T_P is the period corresponding to the peak spectral frequency during a sea state;

When modelling for platform design, we are interested in the extremes of ocean behaviour. The extreme behaviour of a process is typically characterised in one of two standard ways:

- **Block maxima**: under this approach, the data is divided into temporal blocks (in the case of sea states, typically days, months), and the maximum of the process is taken within each block;
- **Peaks over threshold**: here, a threshold is set, and the extremes are defined as the maxima of the process within any continuous period during which it exceeds this threshold.

For ocean sea states, the maxima are most commonly defined in terms of peaks over threshold; any period during which a characteristic exceeds the threshold is classed as a storm, and the largest within the storm is known as the storm-peak value of that characteristic.

In each of these cases, it is possible to derive the limiting distribution of the extreme values (see, for example, Jonathan and Ewans [2013], Coles [2001]) for general distributions; in the Block maxima case, the extremes follow a generalized extreme value (GEV) distribution (for large enough blocks), and in the peaks over threshold case, they are distributed according to a Generalized Pareto (GP) distribution (for large enough thresholds). The GP distribution for H_S at a particular setting of some inputs θ (e.g. location, time) is

$$p\left(H_S\left(\theta\right)|H_S\left(\theta\right) > \mu\left(\theta\right), \xi\left(\theta\right), \sigma\left(\theta\right)\right) = \frac{1}{\sigma\left(\theta\right)}\left[1 + \frac{\xi\left(\theta\right)\left(H_S\left(\theta\right) - \mu\left(\theta\right)\right)}{\sigma\left(\theta\right)}\right]^{-\left(1+\frac{1}{\xi\left(\theta\right)}\right)} . \quad (1.3.3)$$

This distribution for the peaks (over threshold $\mu\left(\theta\right)$) is characterised by a shape parameter $\xi\left(.\right)$ and a scale parameter $\sigma\left(.\right)$. When fitting such an extreme value model, the choice of a suitable threshold $\mu\left(.\right)$ is critical. The fitting of such a covariate-dependent extreme value model is described in, for example, Randell et al. [2015] and Randell et al. [2016], and the paper by Jones et al. [2016] compares different parametrisations of the model. Figure 1.3 (taken from Randell et al. [2015]) shows storm-peak significant wave heights for a location in the Makassar Strait between the islands of Borneo and Sulawesi in Indonesia, observed during the period from August 1956 to July 2012; it is clear from this plot that both of the covariates (direction of arrival and season) have a systematic effect on the distribution of $H_S\left(.\right)$. Since interest lies in the extreme quantiles of the ocean characteristics, a large quantity of data is needed in order to obtain a large enough number of exceedences of a high enough threshold for a model fit. Observation of the real ocean typically only provides a limited amount of data for a small number of locations, and so a common approach is to generate the data for the fitting of the extreme value model using an ocean simulator. To begin with, the ocean simulator is run as a hindcast, using historical atmospheric information to predict historically observed

wave characteristics; the simulator is then tuned to give the best possible match to the sea state characteristics that were actually observed. Once it has been tuned, the simulator is then run using a forecast of future atmospheric characteristics, predicting future wave behaviour; this simulated wave data is then used to fit a covariate-dependent extreme value model to the storm-peak characteristics.

Sources of uncertainty In this problem, the key sources of uncertainty which must be handled are as follows:

- the ocean simulator used to model the historic wave data will typically have a number of non-physical parameters which must be selected so as to give the best representation of the real ocean. We wish to use the observed waves to learn about the range of appropriate settings;
- we then wish to use our uncertainty about these parameters to work out our corresponding uncertainty specification for the simulator at this unknown 'best' setting for all of the other storms that we are modelling;
- since the simulator is a simplified representation of the ocean, there will always be aspects of the real wave behaviour that it fails to capture; we want to use the data that we did observe on the system to learn about the structure of the difference between the simulator and the real ocean, and then compute the implications of this uncertainty for the simulator predictions at all other points;
- when using the simulator to represent the real ocean, uncertainty about the wind field generating the waves explains a substantial amount of the uncertainty about the waves themselves;
- once we have worked out our uncertainty about the wave characteristics at all prediction locations, we aggregate this information and use it to fit an extreme value model of the form (1.3.3); in doing so, we must compute the implications of our uncertainty about $H_S(\theta)$ for our ability to estimate the components of the model.

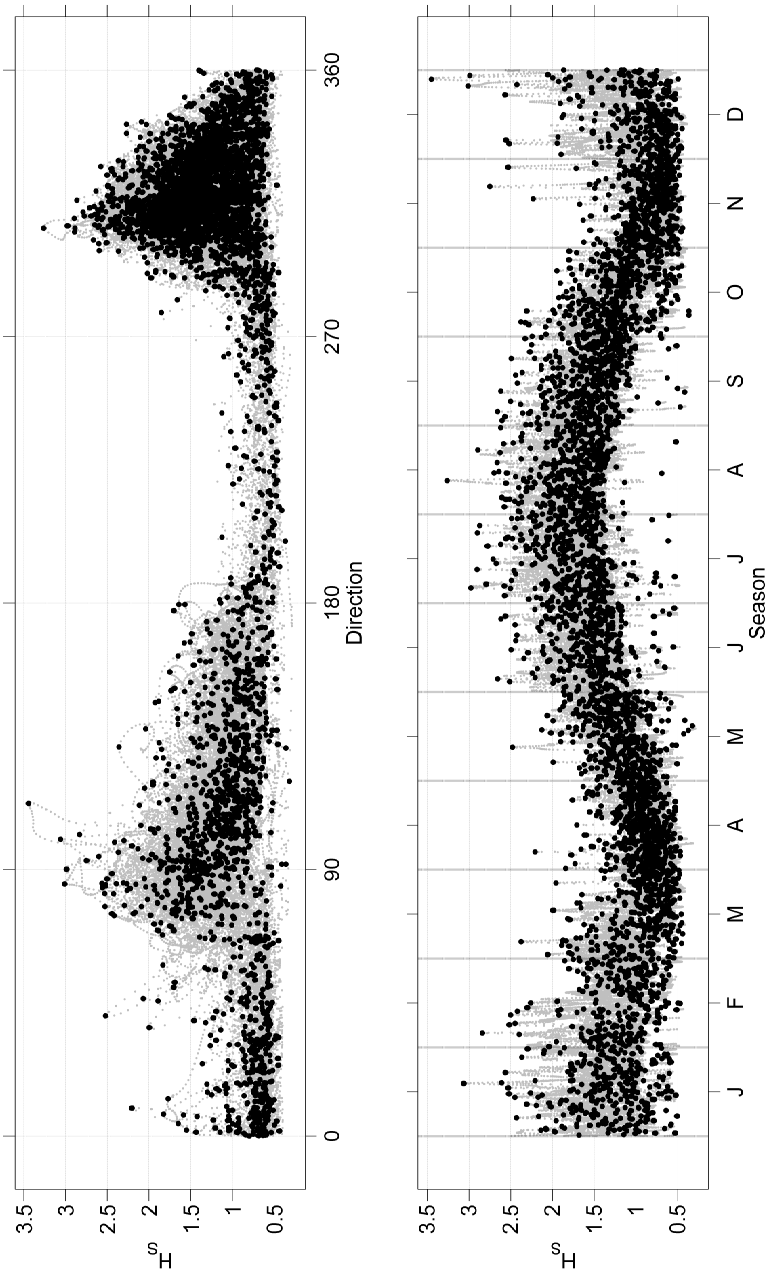


Figure 1.3: Storm-peak significant wave height (black markers) for a location in the Makassar Strait between August 1956 and July 2012, plotted as a function of direction of arrival (top panel) and season (bottom panel); taken from Randell et al. [2015]. The dashed grey lines correspond to storm trajectories.

1.4 Outline of the rest of the thesis

The remainder of this thesis details methodology for handling aspects of the uncertainty detailed in the problem descriptions above, in particular, in relation to experimental design problems for large systems.

In Chapter 2, we outline the Bayesian statistical approach to handling uncertainty, and provide an introduction to relevant material from the literature, which will be called upon in the remainder of the thesis. In Chapter 3, we consider the Bayesian approach to making decisions under uncertainty, again introducing relevant material from the literature; we go on to consider the implications of this for the design of experiments to collect data from the system, and propose a Bayesian treatment of the numerical calculations which we need to carry out in order to generate an optimal design for a given system. In Chapter 4, we present two case studies in which this procedure is applied to experimental design problems.

In Chapter 5, we extend our treatment of the uncertainty in such problems by introducing a framework which allows us to specify beliefs about how our model might evolve in the future, and we examine the implications of this for our existing design calculations. We consider an additional example, in which we link together different development stages for an atmospheric dispersion model. In Chapter 6, we develop a framework for Bayesian analysis of differential equation problems, which can explicitly handle the uncertainty induced by the need to solve the equation numerically in cases where an exact solution to the equation is not known.

Appendix A summarises important aspects of the notation used throughout the thesis, and a selection of the code used to implement the examples in Chapter 4 and Section 6.2 is presented in Appendices D to F. A non-sequential version of the approximate design procedure presented in Chapter 3.4 was published as Jones et al. [2015], and the work presented in Chapters 3 and 4 is currently under review for publication in Technometrics [Jones et al., 2017].

Chapter 2

Bayesian methodology

In this chapter, we introduce some of the Bayesian methodology which will be used throughout the remainder of the thesis. In Section 2.1, we motivate both the fully probabilistic and the Bayes linear forms of Bayesian analysis, starting from the same point, highlighting the similarities and differences between the two, and considering the practicalities of the inference procedure in both instances. Then, in Section 2.2, we give an introduction to emulation, which is simply a special case of the general probabilistic or second-order Bayesian analysis used for the analysis of functions.
In Section 2.3, we review the situations in which emulators are used, and the types of uncertainty which we may handle using them, linking these to the problems discussed in Section 1; then in Section 2.4, we perform some of the calculations necessary to propagate input uncertainty or to do inference using an emulator. Finally, we illustrate some of the elements discussed through application to a simple example in Section 2.5, and through application to a more complex example in Section 2.6.

2.1 Bayesian analysis

During the following chapters, we will have cause to consider two different types of Bayesian analysis; namely, a probabilistic Bayesian analysis and a Bayes linear analysis. As one would perhaps expect, the two types are very similar in nature, and so in the following subsections, we provide motivation for both approaches, starting from the same point in each case, and describe the structure of a typical analysis

for each.

One important point which is worth making straight away is that any specification of beliefs, be they qualitative or quantitative, or about the functional form of a particular model or about a set of quantities within a given model, is entirely subjective and is therefore owned by an individual or a group of individuals who have reached a consensus. This is the *subjective Bayesian* point of view, which will be adopted for the remainder of this thesis. For a more detailed introduction to the subjective Bayesian point of view, see, for example, de Finetti [1975] or Savage [1972].

In what follows, we follow de Finetti (and others; for example, Whittle [1992]) in using expectation as a primitive, where an individual's expectation for a particular quantity X is defined to be the value $\mathrm{E}[X]$ that they would specify if they would incur the following penalty upon realisation of X (for any scalar cost c)

$$L = c(X - \mathrm{E}[X])^2 \,. \tag{2.1.1}$$

If we introduce random quantity A with the further restriction that A is an *event* (i.e. a proposition which is either logically true or false; for example, $X = 3.2$ or 'the postman will visit before 11am today'), and define the indicator function

$$I(A) = \begin{cases} 1 & \text{if } A \text{ occurs,} \\ 0 & \text{else} \end{cases}$$

then the probability of A is defined as $P(A) = \mathrm{E}[I(A)]$. Probabilities are constrained to lie in $[0, 1]$, with 1 indicating certainty of truth of a proposition and 0 indicating certainty of falsehood.

2.1.1 Probabilistic Bayesian inference

A probabilistic Bayesian analysis has, at its core, a specification of the probabilities of all of the different possible outcomes of a given experiment. Even for small problems, it may be extremely challenging to make such a specification, but the reward for doing so is access to a provably coherent set of rules for updating beliefs upon learning the values of certain quantities within the model.

Motivation Jeffrey [2002] considers fair prices for betting slips which pay out upon the occurrence of certain events as a means of proving the basic properties which probability specifications must obey, and of deriving update rules which must be obeyed. For a finite or countably infinite set of incompatible events $A_1, A_2, \ldots$, if we define the event $H = A_1 \vee A_2 \vee \ldots$, then Jeffrey argues that if presented with tickets which pay rewards in probability currency ($P(H)$ if H occurs, $P(A_1)$ if A_1 occurs etc.), then in order to avoid inconsistently valuing the same proposition presented to us in different ways, we must have

$$P(H) = P(A_1) + P(A_2) + \ldots .$$

We may use a similar 'Dutch book' argument to handle relationships between conditional (for example, the probability $P(H|D)$ that H will occur given that D occurs) and joint (for example, the probability $P(H, D)$ that both H and D will occur) probabilities: in a situation where we have a ticket which pays 1 if $H \wedge D$ occurs and $P(H|D)$ if $\neg D$ occurs, it can also be shown that

$$P(H \wedge D) = P(H|D)\, P(D) \ . \tag{2.1.2}$$

If we have a set of events $\{D_i\}$ which form a partition (i.e. exactly one of them must occur), then we can combine these two rules to obtain

$$P(H) = \sum_i P(H|D_i)\, P(D_i) \ .$$

Analysis The results which Jeffrey gives can be generalised to the case of continuous parameters by making appropriate additional assumptions. For a given problem, then, we should perform a Bayesian analysis by introducing a set of assumptions $\mathcal{M}$ (referred to as model assumptions) which specify the functional relationships between a set of parameters $\theta = \{\theta_1, \ldots, \theta_{n_p}\}$; these parameters are assumed to take values in some space Θ. Our model then consists of a probability density function (pdf) $p(\theta|\mathcal{M})$ which generates the probabilities of individual events $\theta \in \chi$ (for some set χ) as follows

$$P(\theta \in \chi|\mathcal{M}) = \int_\chi p(\theta|\mathcal{M})\, d\theta$$

where integration should be replaced by summation in the case of discrete parameters, and our pdfs are normalised so that

$$P\left(\theta \in \Theta | \mathcal{M}\right) = \int_{\Theta} p\left(\theta | \mathcal{M}\right) d\theta = 1 \ . \tag{2.1.3}$$

Using a probabilistic model, we recover the expectation of an individual parameter θ_i as

$$\mathrm{E}\left[\theta_i\right] = \int_{\Theta_i} \theta_i p\left(\theta_i\right) d\theta_i \tag{2.1.4}$$

with similar relations for the variance and the higher-order moments of the distribution. Under any given probability distribution, we can compute the marginal (unconditional) distribution of a set of parameters (indexed by I) by 'integrating out' all others

$$p\left(\theta_I\right) = \int_{\Theta_{\neg I}} p\left(\theta\right) d\theta_{\neg I} \tag{2.1.5}$$

and we can compute the conditional distribution of a set of parameters given the remainder by 'dividing out'

$$p\left(\theta_I | \theta_{\neg I}\right) = \frac{p\left(\theta\right)}{p\left(\theta_{\neg I}\right)} \ .$$

Using this final relation, we can obtain perhaps the most useful probabilistic relationship; if $\theta = \{\alpha, \beta\}$, then as a trivial consequence of the continuous version of the product rule (2.1.2), we have that

$$p\left(\beta | \alpha\right) = \frac{p\left(\alpha | \beta\right) p\left(\beta\right)}{p\left(\alpha\right)} \ . \tag{2.1.6}$$

This relation tells us how we should use data to learn about the world. If, before observing the data, our beliefs about β are summarised through $p\left(\beta\right)$ (known as the prior distribution), and we make a specification $p\left(\alpha | \beta\right)$ for the distribution of α conditional on each possible value of β (often referred to as the likelihood), then if we learn the value of α, we know that our beliefs about β should be updated according to (2.1.6) (where the denominator $p\left(\alpha\right)$ is computed using (2.1.5) and (2.1.2)). This is a powerful and widely applicable result; if α is the outcome of an experiment that we will perform in order to learn about β, and we can specify $p\left(\alpha | \beta\right)$ for all values of α that we might obtain, then (2.1.6) automatically gives us our updated belief state from our prior specification.

Implementation For a continuous parameter θ, the direct specification of a probability density $p(\theta)$ would require us to specify relative density values at an infinite number of parameter settings (and then integrate to enforce the condition (2.1.3)); doing this through explicit consideration of the individual elements of the parameter space Θ is only really possible for discrete problems, and even then, it may present a significant challenge. In practice, therefore, models are built using a small handful of well-known distributional forms.

While a probability distribution should be chosen so that it represents our beliefs about the distribution of a given parameter as faithfully as possible, in practice, the choice of distributions on the basis of their nice computational properties is far more common. Members of the exponential family of distributions are particularly common choices, since they have the very useful property that for certain choices of such distributions for the prior and the likelihood, the posterior distribution will be of the same type as the prior [Diaconis and Ylvisaker, 1979].

Even when using such conjugate distributions to build models, in problems with more than a handful of parameters, we can quickly lose our ability to directly perform the integrals necessary to compute marginal distributions for subsets of the parameters (equation 2.1.5) or to compute moments of functions of parameters (e.g. expectations, variances, covariances; see equation 2.1.4). In recent years, much work has been done towards handling such problems through numerical integration techniques; perhaps the most commonly encountered are Markov-Chain Monte-Carlo (MCMC) methods. These work by generating a Markov chain (a stochastic sequence in which each state is sampled as a function of previous states) in such a way that the set of generated samples is guaranteed to converge in the limit to a set of samples from the required distribution. The Metropolis-Hastings algorithm is the simplest and perhaps the most widely used MCMC method; Robert [2015] gives an introduction, and provides references to more detailed works. A description of a wider range of Monte-Carlo sampling methods is provided in Robert and Casella [1999], and some more advanced methods which exploit gradient information to give better exploration of the distribution are presented in Girolami and Calderhead [2011].

A very large literature now also exists investigating situations in which larger numbers of parameters and a more complex dependency structure are required; graphical models (see, for example, Bishop [2003], Lauritzen and Wermuth [1989], Rue and Held [2005]) provide a useful framework for specifying the dependence structure between parameters, and also for performing the calculations necessary to extract information from the model upon updating using data.

2.1.2 Bayes linear analysis

Instead of building a framework for analysis by specifying a full probability distribution for each of the quantities involved, Bayes linear methods build such a framework starting from only the first- and second-order moments of all of the quantities involved; such a prior specification is much easier to make in almost all situations, and the resulting analysis can often be much simpler to perform.

Motivation For any collection $C = \{C_1, \ldots, C_n\}$ of quantities, we make a prior specification consisting of expectations $\mathrm{E}[C_i]$ and variances $\mathrm{Var}[C_i]$ for each member of the collection, and covariances $\mathrm{Cov}[C_i, C_j]$ for each pair of members. These prior expectations can be interpreted as those which an individual (or a group) would choose under penalty (2.1.1) on realisation of each of the quantities; Goldstein and Wooff [2007] provide a more detailed discussion of some of the issues surrounding such a prior specification.

The fact that a distributional specification is not required already makes such an analysis an appealing prospect in many instances; for example, technical experts in a particular field who have little statistical knowledge may be reluctant to choose a particular distribution to describe a set of parameters, but may be much more willing to give a rough quantification of their beliefs about the means and variances of individual parameters, and any correlations between them. This makes the task of eliciting relevant information about the problem much simpler.

Having made a prior specification for the collection, we consider the effect on our beliefs of learning the values of a given subset of the quantities. Splitting the collection into two subsets, $C = \{B, D\}$, we specify that our updated beliefs about B on

learning D should be a linear combination of the $\{D_i\}$. In the Bayes linear context, we speak of adjusted beliefs; for a single member of B, X, we use $\mathrm{E}_D[X]$ to denote our adjusted expectation for X given observations of the collection D, and we seek coefficients h_i such that

$$\mathrm{E}_D[X] = h_0 + \sum_{i=1}^{n_D} h_i D_i \ .$$

We use the same quadratic penalty as we considered when choosing our prior beliefs; we obtain the adjusted expectation by finding the set of $h = (h_0, h_1, \ldots, h_{n_D})$ which minimises

$$\mathrm{E}\left[\left[X - \sum_{i=0}^{n_D} h_i D_i\right]^2\right]$$

where $D_0 = 1$. It can be shown (see, for example, the book by Goldstein and Wooff) that under this prior specification, choice of linear form and penalty, the resulting adjusted expectation is a function only of the prior specification and the observed values

$$\mathrm{E}_D[X] = \mathrm{E}[X] + \mathrm{Cov}[X, D]\,\mathrm{Var}[D]^{-1}(D - \mathrm{E}[D]) \ .$$

The adjusted variance is then defined as the expectation of the quadratic penalty, and is also simple to compute as a function of the prior moments

$$\mathrm{Var}_D[X] = \mathrm{Var}[X] - \mathrm{Cov}[X, D]\,\mathrm{Var}[D]^{-1}\mathrm{Cov}[D, X] \ .$$

These expressions are easily extended to adjustments for collections of quantities; these expressions are presented in the following paragraph.

Analysis For collections A, B and D, we start by specifying (or eliciting) prior expectations, variances and covariances; the adjusted means, variances and covariances are then computed as follows

Definition 1 *The* ***adjusted expectation*** *of a collection of random quantities B given the observed values of a set of quantities D is given by*

$$E_D[B] = E[B] + Cov[B, D]\ Var[D]^{-1}(D - E[D]) \ .$$

Definition 2 *The* ***adjusted variance*** *of the collection B given D is*

$$Var_D[B] = Var[B] - Cov[B, D]\ Var[D]^{-1}\,Cov[D, B] \ .$$

Definition 3 *The* ***adjusted covariance*** *of the collections B and A given D is*

$$Cov_D[B, A] = Cov[B, A] - Cov[B, D]\, Var[D]^{-1} Cov[D, A] \ .$$

Modelling When using a second-order specification, the whole process of constructing and analysing a model is considerably simpler than when using a fully probabilistic model; we make a prior specification (possibly having elicited this from a domain expert), we observe a subset of the quantities, and we adjust our beliefs about all of the remainder as described. Goldstein and Wooff [2007] (chapters 4 and 5) discuss various diagnostic measures for a Bayes linear model; for example, we may need to revisit our original prior specification in the light of observed data which is outside the range of what we might have expected, or we may wish to assess which components of the observed data contribute the most to our learning about certain other parameters.

A big additional advantage of the Bayes linear framework lies in the fact that all update calculations can be performed in closed-form; no matter how complex the problem under consideration, we will never need to resort to numerical sampling techniques in order to assess the moments of any of our model components. This can lead to huge savings in the computational effort required for some problems. Bayes linear graphical models are useful tools for large problems; these will be considered in more detail in the next section.

2.1.3 Bayes linear graphical models

For a general Bayesian analysis, a graphical model is a useful description of our belief structure; it serves as both a qualitative description of relationships between the different quantities in the problem, and a quantitative framework, which we can use to structure (and potentially reduce the complexity of) the calculations that we need to carry out in order to quantify our beliefs. In a fully Bayesian analysis, the structure of the graph imposes conditional independence relationships between quantities in the problem (which are commonly then exploited in the design of sampling schemes); a Bayes linear graphical model performs the equivalent function for a second-order belief specification.

Directed graphical models A Bayes linear graphical model (Goldstein and Wooff [2007], chapter 10) is a representation of the belief separations between the individual quantities in a problem. A directed acyclic graph (DAG) consists of nodes $\{B_1, B_2, \ldots, B_m\}$, where each node represents a collection of quantities $X_i = \{X_{i1}, X_{i2}, \ldots, X_{in_{B_i}}\}$, with directed edges between some of the nodes. Any two nodes can be connected, subject to the restriction that there are no cycles (i.e. it is not possible to repeatedly visit the same node when tracing out a path along the directions of the edges).

If there is a directed edge from node B_i to node B_j, then we say that B_i is a parent of B_j, and that B_j is a child of B_i; for a general node B_i, we denote its parents by $\mathrm{Pa}(B_i)$ and its children by $\mathrm{Ch}(B_i)$. Based on these definitions, we may construct a (not necessarily unique) ordering of all nodes in the graph by starting at any node with no parents and labelling this as node 1, and then assigning the labels $2, 3, \ldots, m$ sequentially to any node whose parents are already numbered. Any ordering constructed like this has the property that it is not possible to reach any lower-numbered node from a higher-numbered one; the ordering is said to be consistent with the graph.

Based on the above, we define a Bayes linear graphical model as follows [Goldstein and Wooff, 2007]

Definition 4 *A model is a* ***directed (second-order) graphical model*** *if, when $B_1, \ldots, B_m$ is a consistent ordering on the nodes, then for each k, node B_k is separated by its parent nodes from all predecessor nodes in the list; we write*

$$B_k \perp B(k-1)/Pa(B_k) \tag{2.1.7}$$

where $B(j) = \{B_1, \ldots, B_j\}$

If the nodes B_j and B_k are separated by the node B_i ($B_j \perp B_k/B_i$ in the above notation), this implies that the collection of quantities represented by B_i is Bayes linear sufficient for B_k for adjusting B_j; this is the Bayes linear analogue of the conditional independence property in the fully probabilistic case, and implies that B_k will not provide any further information about B_j after adjustment by B_i. In the Bayes linear case, we speak of belief separation, rather than of conditional independence.

Computations on the graph Once we have specified a graph which captures all of our qualitative beliefs about the structure of the model, we can use the graph as a tool to quantify aspects of our belief structure. For a general problem, we must make a full second-order belief specification for all quantities, comprising expectations and variances for each quantity, and covariances between pairs of quantities; however, if we can represent the structure of our model using a DAG, we only need to specify a subset of these moments, and the structure of the graph will determine the rest. If for any sets of quantities B_i, B_j, B_k, we have that $B_k \perp B_j / B_i$, then this implies that

$$\mathrm{Cov}\left[B_k, B_j\right] = \mathrm{Cov}\left[B_k, B_i\right] \mathrm{Var}\left[B_i\right]^{-1} \mathrm{Cov}\left[B_i, B_j\right]$$

so we may determine the covariance between B_k and B_j in terms of the covariance of each with B_i and the variance $\mathrm{Var}\left[B_i\right]$. Using the property (2.1.7) of the graphical model, we know that for any consistent ordering of the nodes, a node is separated from all lower-numbered nodes in the graph by its parent set; combining these two properties, for nodes B_j and B_k, with $j < k$, we have

$$\mathrm{Cov}\left[B_k, B_j\right] = \mathrm{Cov}\left[B_k, \mathrm{Pa}\left(B_k\right)\right] \mathrm{Var}\left[\mathrm{Pa}\left(B_k\right)\right]^{-1} \mathrm{Cov}\left[\mathrm{Pa}\left(B_k\right), B_j\right] \ . \tag{2.1.8}$$

This means that, for a graphical model, we must only make a limited prior specification, consisting of expectations and variances for all nodes, and covariances between nodes corresponding to the edges in the graph; given this specification, all remaining covariances (which do not correspond to edges in the graph) are determined by the graph structure through (2.1.8).

Junction trees For a general DAG, repeatedly exploiting the result (2.1.8) allows us to compute the covariance between any pair of nodes. In general, however, this procedure can be quite complex, owing to the fact that there may be many different paths between a pair of nodes, all of which must be accounted for. For a particular problem, we can simplify the calculations by imposing a simple graph structure; if the graph is a tree (a graph in which there is only one path between each pair of nodes), then we must only compute covariances along a chain of parent nodes, and minor local deviations from this form will generally not make the calculations

too onerous. In many instances, our actual beliefs will suggest a structure which is considerably more complex than this, and so we may be unwilling to make such a big simplification.

It is, however, relatively simple to reduce a general directed graph to an undirected tree; if we can find a way to compute covariances between nodes using the undirected graph, then this will greatly reduce the complexity of the procedure. We start down this path by defining the undirected moral graph associated with a DAG.

Definition 5 *The **moral graph** associated with a particular DAG is formed by:*

(i) 'marrying' unmarried parents by drawing an (undirected) arc between any two unconnected nodes with a common child;

(ii) dropping the arrows from all directed edges.

Goldstein and Wooff [2007] (chapter 10) discuss the properties of general undirected graphical models in some detail. We now use the procedure outlined in the book by Goldstein and Wooff to find the (non-unique) junction tree associated with a DAG.

Definition 6 *A **Junction tree** can be constructed from a general DAG through the following procedure:*

(i) Create the moral graph associated with the DAG;

(ii) Triangulate the graph- add edges to ensure that there are no cycles of length 4 *or more without a chord;*

(iii) Carry out a maximum cardinality search: arbitrarily pick a node and label it as node 1*; then, at steps* $k = 2, \ldots, m$*, we label as node* k *the node with the greatest number of labelled neighbours;*

(iv) Find and label the cliques:

- *the cliques* $\{Q_i\}$ *are the maximal sets of nodes which are all joined to each other;*
- *these cliques are ordered according to the highest-numbered node within them;*

(v) Create the junction tree:

- *the nodes of the tree are the cliques;*
- *each clique Q_i is joined to at most one of the lower-numbered cliques $\{Q_j\}$, $j < i$ by imposing an edge between it and any one of the earlier cliques which contains the intersection between Q_i and all nodes in all lower-numbered cliques.*

The proof that this algorithm generates a junction tree is provided in, for example, Lauritzen [1996]; the properties of a junction tree are explored in more detail in Goldstein and Wooff [2007]. The junction tree generated through the above algorithm is an undirected graph with the property that there is only one path between any pair of nodes.

Using this construction, we can compute covariances between any nodes in adjacent cliques as follows: if cliques Q_i and Q_j are neighbours in the junction tree, and we denote the intersection between these cliques by $A = \mathrm{Q}_i \cap \mathrm{Q}_j$, then the covariance between nodes $B_k \in \mathrm{Q}_i$ and $B_l \in \mathrm{Q}_j$ is

$$\mathrm{Cov}\left[B_k, B_l\right] = \mathrm{Cov}\left[B_k, A\right] \mathrm{Var}\left[A\right]^{-1} \mathrm{Cov}\left[A, B_l\right] \ . \tag{2.1.9}$$

We can use the relation (2.1.9) to compute the covariance between any pair of nodes on the graph; we simply propagate the covariance through the cliques by sequentially computing covariances between clique intersections along the tree.

2.2 Bayesian analysis for functions

A Bayesian analysis for a function is simply a particular case of the standard probabilistic or Bayes linear analysis; however, it is one which has received much attention because of its wide range of possible applications. Since this type of model will be used repeatedly in the remainder of this thesis, we explore it here in detail. In the following subsections, we consider two possible prior uncertainty specifications that can be made for functions, and present the calculations for updating beliefs in both cases; a fully probabilistic prior specification leads to a Gaussian process model,

which is outlined in Section 2.2.1, and a second-order prior specification produces a Bayes linear emulator, discussed in Section 2.2.2. In Section 2.3, we outline some of the modelling problems that can be addressed using this type of model, and in Section 2.4, we detail some of the calculations that we will need to perform to achieve these modelling goals.

2.2.1 Gaussian Processes

Formally, a Gaussian process is a (possibly infinite) collection of quantities, any finite-dimensional subset of which has a multivariate Gaussian distribution (see, for example, Rasmussen and Williams [2006]); we write

$$r(\theta) \sim \mathcal{GP}(0, k(\theta, \theta'))$$

to denote that the scalar function $r(.)$ is a mean-zero Gaussian process, where covariances between function values at different input settings are determined by the covariance function $k(\theta, \theta')$

$$\mathrm{Cov}[r(\theta), r(\theta') | \lambda] = k(\theta, \theta' | \lambda)$$

where λ are parameters of the covariance function, which are assumed fixed for the purposes of this discussion (Section 2.2.3 discusses the determination of λ); we suppress the dependence of the covariance on λ for the remainder of this section.
When building a model, a Gaussian process is often combined with a linear regression term; the regression surface is used to capture any global trends in the function value, and the Gaussian process acts as a correlated residual, picking up any systematic local deviations from these global trends. Our model for a general, scalar function $f(.)$ is

$$f(\theta) = g(\theta)^{\mathrm{T}} \beta + r(\theta) \tag{2.2.10}$$

where $g(\theta) = (g_1(\theta), \ldots, g_{n_g}(\theta))^{\mathrm{T}}$ is a vector of known basis functions, and β is a corresponding vector of weighting parameters, about which we are uncertain. We specify our prior beliefs about β through a probability distribution $p(\beta)$; β and $r(.)$ are assumed to be a priori independent of each other. The basis functions $g(.)$ are either specified using our prior knowledge about the function, or determined from

the data; for further discussion on the selection of a suitable basis, see Section 2.2.3. If we further specify a Gaussian prior distribution for the regression parameters, $\beta \sim \mathcal{N}(\mu_\beta, V_\beta)$, then we may exploit the Gaussianity of finite-dimensional collections of values of $r(.)$ to derive a closed-form relationship between a set of observed function values and our corresponding updated beliefs about the function at any input θ. We observe the function at the input points $\Theta = \{\theta_1, \ldots, \theta_n\}$, obtaining data $F = (F_1, \ldots, F_n)^{\mathrm{T}}$, where $F_j = f(\theta_j) + \epsilon_j$ and the ϵ_j are mean-zero measurement error terms, with $\epsilon_j \sim \mathcal{N}(0, \sigma^2)$ independently of all other components and for each observation. By marginalising over the β and conditioning [Rasmussen and Williams, 2006], we obtain another Gaussian process summarising our posterior beliefs about $f(\theta)$ given F

$$f(\theta)\,|F \sim \mathcal{GP}\left(\hat{m}(\theta), \hat{k}(\theta, \theta')\right)$$

where

$$\begin{aligned}\hat{m}(\theta) &= g(\theta)\,\hat{\beta} + d(\theta)^{\mathrm{T}} K^{-1}(F - G\hat{\beta}) \\ \hat{k}(\theta, \theta') &= k(\theta, \theta') - d(\theta)^{\mathrm{T}} K^{-1} d(\theta') + R(\theta)^{\mathrm{T}} \hat{V}_\beta R(\theta')\end{aligned}$$

where $d(\theta) = (d_1(\theta), \ldots, d_n(\theta))^{\mathrm{T}}$ is a vector with elements $d_i(\theta) = k(\theta, \theta_i)$, K is a covariance matrix with elements $K_{ij} = k(\theta_i, \theta_j) + \sigma^2 I_{ij}$, G is a design matrix with elements $G_{ij} = g_j(\theta_i)$, and

$$R(\theta) = g(\theta)^{\mathrm{T}} - G^{\mathrm{T}} K^{-1} d(\theta)$$

and

$$\begin{aligned}\hat{V}_\beta &= \left[{V_\beta}^{-1} + G^{\mathrm{T}} K^{-1} G\right]^{-1} \\ \hat{\beta} &= \hat{V}_\beta \left[{V_\beta}^{-1} \mu_\beta + G^{\mathrm{T}} K^{-1} F\right].\end{aligned}$$

The above calculations were all carried out conditionally on fixed values of any parameters λ present in the covariance function; of course, in practice, we will also be uncertain about the settings of these parameters which are appropriate for a particular analysis. Under a fully Bayesian framework, we should specify our prior beliefs $p(\lambda)$ about these parameters, and then compute marginal posterior beliefs

about $f(.)$ by integrating over these beliefs; for most useful choices of covariance function, however, this is not possible analytically. For a fully Bayesian analysis, then, we would need to use a numerical integration procedure (typically an MCMC scheme).

A more common and practical approach is to select the parameters of the covariance function using a maximum likelihood procedure and then carry out the remainder of the analysis using these fixed values. In Section 2.2.3, we discuss a maximum likelihood procedure based on leave-one-out cross-validation, in which we maximise the predictive likelihood for each point based on a fit to the remainder.

It is relatively simple to extend the above analysis to cases where $f(.)$ is a multi-output function; we simply vectorise all of the quantities. To avoid cluttering the the notation, we do not consider multivariate function models in the probabilistic case in this thesis; we do, however, consider multi-output second-order function models in Section 2.2.2. For a treatment of the multivariate Gaussian process case, see, for example, Rougier [2008], Conti and O'Hagan [2010] or Fricker et al. [2013]. The paper by Rougier also considers how we might impose and exploit a separable covariance structure in order to efficiently invert the matrix K.

2.2.2 Second-order analysis

A second-order analysis for functional data begins in exactly the same way as a fully probabilistic analysis, through specification of a model for the function as the sum of a regression term and a correlated residual process as in (2.2.10); in this context, however, we take the same approach as Williamson [2010] in introducing index notation and considering multi-output functions

$$f_i(\theta) = \beta_{ip} g_p(\theta) + r_i(\theta) \tag{2.2.11}$$

where we use the Einstein summation convention that repeated indices are summed over. In the Bayes linear context, we make a second-order prior specification for all of the quantities on the right-hand side (i.e. $\mathrm{E}[\beta_{ip}]$, $\mathrm{Cov}[\beta_{ip}, \beta_{kq}]$ and $\mathrm{Cov}[r_i(\theta), r_k(\theta')]$, again making the assumptions that β and $r(\theta)$ are uncorrelated and that $r(\theta)$ has

zero prior mean); we can then use noise-corrupted measurements $F_{ij} = f_i(\theta_j) + \epsilon_{ij}$ of the function at a known set of locations $\Theta = \{\theta_1, \ldots, \theta_j, \ldots, \theta_n\}$ to adjust our beliefs about these quantities, and about the function value at unobserved inputs. The expectations of and covariances between function observations can be calculated using our prior beliefs

$$\begin{aligned} \mathrm{E}[F_{ij}] &= \mathrm{E}[\beta_{ip}] G_{pj} \\ \mathrm{Cov}[F_{ij}, F_{kl}] &= G_{pj}\mathrm{Cov}[\beta_{ip}, \beta_{kq}] G_{ql} + \mathrm{Cov}[r_i(\theta_j), r_k(\theta_l)] \\ &\quad + \mathrm{Cov}[\epsilon_{ij}, \epsilon_{kl}] \end{aligned}$$

where $G_{pj} = g_p(\theta_j)$ is the usual basis matrix, and we have assumed that the noise terms ϵ_{ij} are uncorrelated with the other components. We use $\mathrm{Var}[F]$ to denote the four-dimensional object with elements $\mathrm{Var}[F]_{ijkl} = \mathrm{Cov}[F_{ij}, F_{kl}]$, and we compute the elements of its inverse as follows: we create the vector $\tilde{F}$ by stacking the data so that $\tilde{F}_{i+(j-1)n_f} = F_{ij}$, and set

$$\mathrm{Var}[F]^{-1}_{ijkl} = \mathrm{Var}\left[\tilde{F}\right]^{-1}_{(i+(j-1)n_f),(k+(l-1)n_f)}$$

where $\mathrm{Var}[F]^{-1}_{ijkl}$ is the $(ijkl)^{\text{th}}$ element of $\mathrm{Var}[F]^{-1}$. We can compute the adjusted moments of the parameters β and the process $r(\theta)$ as follows (see, for example, Williamson [2010], Craig et al. [2001], Vernon et al. [2010] for examples of similar calculations)

- $\mathrm{E}_F[\beta_{ij}]$:

$$\begin{aligned} \mathrm{E}_F[\beta_{ij}] &= \mathrm{E}[\beta_{ij}] + \mathrm{Cov}[\beta_{ij}, F_{kl}] \mathrm{Var}[F]^{-1}_{klpq}\left[F_{pq} - \mathrm{E}[F_{pq}]\right] \\ &= \mathrm{E}[\beta_{ij}] + \mathrm{Cov}[\beta_{ij}, \beta_{kr}] G_{rl}\mathrm{Var}[F]^{-1}_{klpq}\left[F_{pq} - \mathrm{E}[F_{pq}]\right] \end{aligned}$$

- $\mathrm{Cov}_F[\beta_{ij}, \beta_{kl}]$:

$$\begin{aligned} \mathrm{Cov}_F[\beta_{ij}, \beta_{kl}] &= \mathrm{Cov}[\beta_{ij}, \beta_{kl}] \\ &\quad - \mathrm{Cov}[\beta_{ij}, \beta_{pr}] G_{rq}\mathrm{Var}[F]^{-1}_{pqst} G_{ut}\mathrm{Cov}[\beta_{su}, \beta_{kl}] \end{aligned}$$

- $\mathrm{E}_F[r_i(\theta)]$:

$$\begin{aligned} \mathrm{E}_F[r_i(\theta)] &= \mathrm{E}[r_i(\theta)] + \mathrm{Cov}[r_i(\theta), F_{kl}] \mathrm{Var}[F]^{-1}_{klpq}\left[F_{pq} - \mathrm{E}[F_{pq}]\right] \\ &= \mathrm{Cov}[r_i(\theta), r_k(\theta_l)] \mathrm{Var}[F]^{-1}_{klpq}\left[F_{pq} - \mathrm{E}[F_{pq}]\right] \end{aligned}$$

- $\mathrm{Cov}_F\left[r_i\left(\theta\right), r_j\left(\theta'\right)\right]$:

$$\begin{aligned}\mathrm{Cov}_F\left[r_i\left(\theta\right), r_j\left(\theta'\right)\right] &= \mathrm{Cov}\left[r_i\left(\theta\right), r_j\left(\theta'\right)\right] \\ &\quad - \mathrm{Cov}\left[r_i\left(\theta\right), r_k\left(\theta_l\right)\right] \mathrm{Var}\left[F\right]^{-1}_{klpq} \mathrm{Cov}\left[r_p\left(\theta_q\right), r_j\left(\theta'\right)\right]\end{aligned}$$

- $\mathrm{Cov}_F\left[\beta_{ij}, r_k\left(\theta\right)\right]$

$$\mathrm{Cov}_F\left[\beta_{ij}, r_k\left(\theta\right)\right] = -\mathrm{Cov}\left[\beta_{ij}, \beta_{pr}\right] G_{rq} \mathrm{Var}\left[F\right]^{-1}_{pqst} \mathrm{Cov}\left[r_s\left(\theta_t\right), r_k\left(\theta\right)\right]$$

The adjusted moments of the function $f\left(.\right)$ at a new input θ can now be computed by re-combining these elements using simple properties of adjusted expectation and covariance [Goldstein and Wooff, 2007]. Our adjusted expectation is

$$\mathrm{E}_F\left[f_i\left(\theta\right)\right] = \mathrm{E}_F\left[\beta_{ik}\right] g_k\left(\theta\right) + \mathrm{E}_F\left[r_i\left(\theta\right)\right] \tag{2.2.12}$$

and our adjusted covariances between function values at new inputs are

$$\begin{aligned}\mathrm{Cov}_F\left[f_i\left(\theta\right), f_j\left(\theta'\right)\right] =& g_k\left(\theta\right) \mathrm{Cov}_F\left[\beta_{ik}, \beta_{jl}\right] g_l\left(\theta'\right) + \mathrm{Cov}_F\left[r_i\left(\theta\right), r_j\left(\theta'\right)\right] \\ &+ g_k\left(\theta\right) \mathrm{Cov}_F\left[\beta_{ik}, r_j\left(\theta'\right)\right] + \mathrm{Cov}_F\left[r_i\left(\theta\right), \beta_{jl}\right] g_l\left(\theta'\right)\end{aligned} \tag{2.2.13}$$

Again, our prior specification $\mathrm{Cov}\left[r_i\left(\theta\right), r_j\left(\theta'\right)\right] = k_{ij}\left(\theta, \theta' | \lambda\right)$ for the covariance structure of $r\left(.\right)$ takes the form of a covariance function which depends in a complex manner on parameters λ. The above analysis is all carried out conditionally on a fixed value of λ, and procedures for determining a suitable value are discussed in Section 2.2.3.

2.2.3 Covariance functions

One of the main issues encountered when trying to fit both Gaussian process models and second-order emulators is in the selection of a covariance function and the determination of its parameters. In both models, the covariance function determines the marginal variance of the residual process output $r_i\left(\theta\right)$ at input θ and the degree of correlation between residual function outputs $r_i\left(\theta\right)$ and $r_j\left(\theta'\right)$ at different

input locations θ and θ'. A common simplification is to decouple the inter-output covariance structure from the correlation induced as a function of θ, as

$$\begin{aligned} \mathrm{Cov}\left[r_i\left(\theta\right), r_j\left(\theta'\right)\right] &= k_{ij}\left(\theta, \theta'\right) \\ &= V_{ij} c\left(\theta, \theta'\right) \end{aligned} \tag{2.2.14}$$

where V_{ij} is the marginal covariance between outputs $r_i\left(.\right)$ and $r_j\left(.\right)$ (with V_{ii} the marginal variance of an individual output i), and $c\left(\theta, \theta'\right)$ is a correlation function, which determines the degree of correlation between the pair of outputs evaluated at θ and θ'. In order to ensure that the resulting covariance matrices are positive-definite, we require that $c\left(.,.\right)$ is a positive-definite function, and that V is a positive-definite matrix. Two common choices of correlation function are the squared-exponential and the Matèrn forms, both of which are outlined below.

Squared exponential This is simply an un-normalised Gaussian form

$$c\left(\theta, \theta'\right) = \exp\left[-\frac{1}{2}\left(l\left(\theta, \theta'\right)\right)^2\right]$$

where

$$l\left(\theta, \theta'\right) = \left[(\theta - \theta')^{\mathrm{T}} \Lambda (\theta - \theta')\right]^{1/2}$$

is a scaled distance between θ and θ', and Λ is a positive-definite matrix. The eigenvectors of the matrix Λ determine the (orthogonal) principal directions of correlation imposed, and the rates of decay of the correlation along these directions are governed by the corresponding eigenvalues. This form is popular because it is simple to work with (for much the same reasons that the Gaussian distribution is easy to work with). In particular, choosing a squared exponential covariance can greatly simplify a number of the more complex calculations that are described in Section 2.4; further details are given there and in appendix B.

Matèrn This correlation function also depends on the same transformed distance between points, but in a more complex manner

$$c\left(\theta, \theta'\right) = \frac{1}{2^{\nu-1}\Gamma\left(\nu\right)} \left(l\left(\theta, \theta'\right)\right)^{\nu} K_{\nu}\left(l\left(\theta, \theta'\right)\right) \ .$$

As well as the matrix of scaling parameters Λ for the distance measure, the Matèrn correlation is also parametrized by a roughness measure ν. This second parameter controls the differentiability of the covariance function, and thereby the smoothness of the process; this makes the Matèrn correlation function much more flexible than the squared exponential, which is useful in cases where the infinite-differentiability of the latter is clearly not justified. However, this flexibility needs to be traded off against the need to select an additional parameter, and the additional complexity that this choice may entail for some of the calculations in Section 2.3.

Determining the parameters In order to perform a fully Bayesian analysis for a function, we need to specify a suitable model (Gaussian process or Bayes linear), and then use the data to derive updated beliefs about the coefficients β, the inter-output covariance matrix V and the correlation function parameters λ before using these posterior beliefs to make predictions about function values at unobserved input locations. However, in practice, this is a very challenging task, as the dependence of the covariance function on its parameters is typically complex, and as such, no closed-form update rules exist. In the fully Bayesian context, we would need to resort to sampling methods to obtain parameter estimates for the correlation function parameters, and then to characterise the predictive distribution for $f(\theta)$, which would significantly complicate the analysis, and remove the possibility of a closed-form expression for predictions at new input locations.

A much more common approach is to fix V and λ using empirical procedures. For the correlation parameters we may use a cross-validation procedure: when fitting an emulator, we divide the data into a 'training' dataset and a 'test' dataset. The training data is used to fit the model, and then the model is used to predict the test data; the quality of this prediction is then assessed for different settings of the covariance function parameters, and the best such setting (by some suitable criterion) is then fixed and used in all predictions for new function values.

In the Gaussian process case, the approach that is most commonly taken is to numerically maximise the predictive log-likelihood of the test points over the covariance function parameters; for test input set $\hat{\Theta} = \{\hat{\theta}_1, \ldots, \hat{\theta}_{n_c}\}$, and test data $\hat{F}_j = f\left(\hat{\theta}_j\right)$,

we have

$$l(\lambda) = -\frac{1}{2}\log\left(\left|\hat{K}^{-1}\right|\right) - \frac{1}{2}\left(\hat{F}-\hat{m}\right)^{\mathrm{T}}\hat{K}^{-1}\left(\hat{F}-\hat{m}\right)$$

where $\hat{m} = (\hat{m}_1, \ldots, \hat{m}_{n_c})^{\mathrm{T}}$ with $\hat{m}_i = \hat{m}\left(\hat{\theta}_i|\lambda\right)$, and $\hat{K}$ is the predictive covariance matrix with $\hat{K}_{ij} = \hat{k}\left(\hat{\theta}_i, \hat{\theta}_j|\lambda\right)$. In cases where data is scarce, and we cannot sacrifice it to being part of the test set, leave-one-out cross validation is a popular variant of this procedure; each individual data point is left out in turn, and is predicted from a Gaussian process fit to the rest. Rasmussen and Williams [2006] show how the likelihood for the leave-one-out procedure can be computed in such a way that we only need to invert the data covariance matrix once, making this an extremely efficient procedure.

In the Bayes linear case, a common choice of criterion is the Mahalanobis distance; assuming multi-output functions again, we have

$$M(\lambda) = \left(\hat{F}_{ij} - \mathrm{E}_F\left[\hat{F}_{ij}|\lambda\right]\right)\mathrm{Var}_F\left[\hat{F}|\lambda\right]^{-1}_{ijkl}\left(\hat{F}_{kl} - \mathrm{E}_F\left[\hat{F}_{kl}|\lambda\right]\right).$$

Low values of this quantity indicate a good prediction for the test data, and so we minimise over λ.

Fast inversion & memory management When fitting a Gaussian process or second-order emulator, the most computationally expensive and memory-hungry step is the calculation, inversion and storage of the inverse data covariance matrix, $\mathrm{Var}[F]^{-1}$. If our model has n_f outputs and we have access to n runs, then $\mathrm{Var}[F]$ is an $(n_f \times n) \times (n_f \times n)$ matrix, and for moderately high values of n_f and n, it is easy for the inversion of this matrix to become unmanageable.

Where we anticipate that this will be a problem, we can structure the covariance function and the design of the simulator runs in such a way that we can take advantage of the Woodbury matrix inversion lemma [Rougier, 2008]; if we define the reshaped variances

$$\begin{aligned}
U_{i+(k-1)n_f, j+(l-1)n_f} &= \mathrm{Var}\left[F_{ik}, F_{jl}\right] \\
B_{i+(k-1)n_f, j+(l-1)n_f} &= \mathrm{Var}\left[\beta_{ik}, \beta_{jl}\right] \\
R_{i+(k-1)n_f, j+(l-1)n_f} &= \mathrm{Var}\left[r_i\left(\theta_k\right), r_j\left(\theta_l\right)\right]
\end{aligned}$$

then the variance of the vectorised simulator data is

$$U = \tilde{G} B \tilde{G}^{\mathrm{T}} + R$$

where $\tilde{G} = G^{\mathrm{T}} \otimes I_{n_f}$. If we apply the Woodbury matrix inversion lemma to this form, then we obtain

$$U^{-1} = R^{-1} - (R^{-1}\tilde{G})(B + \tilde{G}R^{-1}\tilde{G})^{-1}(R^{-1}\tilde{G})^{\mathrm{T}} \ . \tag{2.2.15}$$

If R has a Kronecker product structure, then we can compute the inverse U^{-1} much more quickly using this expression; the inverse of the matrix $B + \tilde{G}R^{-1}\tilde{G}$ is of dimension $(n_f n_g) \times (n_f n_g)$, which is usually much smaller than the original variance matrix, and the inverse of R can be computed as

$$R^{-1} = \Sigma_\theta^{-1} \otimes V^{-1} \ .$$

If there is further product structure in the matrix R, then we can compute the inverse even more efficiently by exploiting this structure too. We recover the inverse object required for the update equations in Section 2.2.2 as

$$\mathrm{Var}\,[F]^{-1}_{ikjl} = \left[U^{-1}\right]_{i+(k-1)n_f, j+(l-1)n_f} \ .$$

If U is so large that even carrying it around inside the update code presents memory issue, then the structure of the inverse (2.2.15) can also be used to reduce the memory requirements. Instead of holding U^{-1} in memory, we store the components of the product, and we also compute and store $(R^{-1}\tilde{G})$ (which is of dimension $(n_f n) \times (n_f n_g)$); any future products of U^{-1} with the covariance can be computed using the product structure, without having to compute the whole matrix, and any products of U^{-1} with the basis terms can be computed using $(R^{-1}\tilde{G})$, which is much smaller than the full matrix.

2.2.4 Model fitting procedure

We detail the procedure that we will use to fit emulators in the remainder of the thesis, combining all of the elements discussed above:

- We begin by using a small subset $\tilde{F}$ of the data to fit the regression surface, in the absence of the residual term. If we are unsure as to which basis functions to choose, then we first carry out the regression using multiple different basis choices and compare the parameter estimates which we obtain; having settled on a basis, the parameter estimates from the regression are used to fix the prior moments $\mathrm{E}\left[\beta_{ij}\right]$ and $\mathrm{Cov}\left[\beta_{ij}, \beta_{kl}\right]$.

- In order to fix the inter-output covariance matrix V, we use the fitted regression surface to compute the residuals $R_{ij} = \tilde{F}_{ij} - \mathrm{E}\left[\beta_{ip}\right] G_{pj}$, and fix the components of V to the empirical residual covariances

$$V_{ij} = \frac{1}{n_{\tilde{F}}} \sum_{k} \left(R_{ik} - \bar{R}_i\right) \left(R_{jk} - \bar{R}_j\right)$$

  where $\bar{R}_i = \frac{1}{n_{\tilde{F}}} \sum_k R_{ij}$.

- Once the marginal covariances have been fixed, we use a cross-validation procedure (as outlined at the end of Section 2.2.3) to determine the correlation parameters λ; we fix them to a value $\hat{\lambda}$ which maximises a suitable criterion, and then use the covariance function $\mathrm{Cov}\left[r\left(\theta\right), r\left(\theta'\right)\right] = \mathrm{Cov}\left[r\left(\theta\right), r\left(\theta'\right) | \hat{\lambda}\right]$ for the remainder of the analysis.

2.3 Applications

Section 2.2 details how to fit a model comprising a regression component and a correlated residual process to a set of observations of a function at known input locations, in order to use this model to make predictions of the function at input settings where it has not yet been sampled. While predicting the function may be of interest in its own right, we are often more interested in making statements about other quantities which are related to this function.

Often, the function which is being modelled is itself a representation of a particular system, commonly one which has a long run time, with the model being used as a quick approximation to its output. In this situation, we are interested in being able to use our model to make statements about the system, or to use data from

the system to tune our model; the different aspects of this problem are considered in detail in Section 2.3.1.

Another common situation is that in which we are not interested in the function itself, but in integrals or derivatives thereof; for example, we may want to translate our beliefs about a function into beliefs about its expectation with respect to the distribution of one of its inputs. Emulators make this task an extremely simple one; we consider previous work in this area in Section 2.3.2.

2.3.1 Uncertainty analysis for complex functions

As a result of rapid advances in the capabilities of computers in the latter part of the 20^{th} century, there has been a correspondingly rapid increase in the scope and scale of models that are used to study various phenomena; scientists increasingly find themselves in situations where their knowledge of the behaviour of a system leads them to postulate a model which has a very large number of input and/or output parameters and takes a significant amount of time to run for any given setting of these variables. For example, in climate science, even low resolution models can take days to run on a cluster (see, for example, Smith et al. [2008], Williamson et al. [2013]), and still contain many hundreds of input parameters which must be specified for each run of the model.

These models are also generally deterministic (a given setting of the input parameters will always lead to the same output); this presents an additional set of challenges when we come to link the model to the system which it is supposed to represent. Frequently, we will be uncertain about both a subset of the input parameters, and the relationship between the model output and the system under any given setting at which it is run.

In this context, we introduce a distinction between the model that we postulate for a system, and the computer simulator that we use to actually use to generate numerical values for attributes of our system; the relationship between the two is the following [Vernon et al., 2010]

$$\text{Simulator} = \text{Model} + \text{Treatment} .$$

That is, the model is the abstract mathematical description of a system (possibly only implicitly defined– see Section 1.1.1 for an example), whereas the simulator is the actual implementation which allows us to predict aspects of the system. The 'Treatment' component of the simulator may, for example, consist of a particular set of decisions about the domain discretization that we use to solve the governing equations. The model may sometimes be sufficiently simple that we can implement it exactly, without having to make any decisions about its numerical treatment (for example, the Gaussian plume model in Section 1.1.3); in this situation, the model and the simulator are the same object.

Bayesian analysis of such models conveys great advantages; an emulator (i.e. a stochastic representation of the form outlined in Sections 2.2.1 or 2.2.2) can be fitted to a small number of runs on the full computer simulator, and then used to predict outputs at as-yet unobserved input settings in a fraction of the time it would take to run the simulator again. We can also use this probabilistic or second-order framework to specify our beliefs about the discrepancy between the simulator outputs and the real values that the system takes, and use input uncertainty to generate a corresponding uncertainty specification for the system value. This type of analysis is generally referred to as a Bayesian uncertainty analysis.

Belief specification We build up our belief specification about the data observed on the system at given input settings by considering uncertainties arising from a number of different sources. We adopt the same specification as used by Kennedy and O'Hagan [2001], Vernon et al. [2010] and others; we assume that the observations z which we make on the system are noise-corrupted measurements of underlying system values $y(.)$ measured at known settings of some system inputs b

$$z_{ij} = y_i(b_j) + \epsilon_i(b_j) \tag{2.3.16}$$

where $y_i(.)$ is the i^{th} system output, and $\epsilon(.)$ is a measurement noise process, for which we make an appropriate prior specification. $y(.)$ and $\epsilon(.)$ are assumed to be independent of each other; while we may not believe this to be the case in some instances (for example, the measurement error is proportional to the system value in many real applications), we will generally make this assumption in order to retain

tractability of later calculations. We denote our simulator by $f(.)$; this simulator is built so as to mimic the behaviour of the system, but is assumed to be an imperfect representation of it.

We also assume that there is an additional set of simulator inputs a which must be specified in order to run the simulator, but which do not necessarily have physical analogues, and are not measured. We make a 'best input' assumption for these parameters (see, for example, Craig et al. [2001]); that is, we assume that there exists a setting a^* of these parameters such that running the simulator at this point will provide all of the information available in $f(.)$ about $y(.)$.

We specify the following relationship between the simulator and the system

$$y_i(b) = \hat{f}_i(b) + \delta_i(b) \tag{2.3.17}$$

where $\hat{f}(b) = f(a^*, b)$ is the simulator evaluated at its best input, and $\delta(.)$ is the discrepancy between the simulator and the system value at this point. We also specify that $\hat{f}(.)$ and $\delta(.)$ are a priori independent.

If the simulator itself is a complex function and has a long run time, then we also fit an emulator as a surrogate; this emulator has the same form as given in (2.2.11), where the full emulator input set is $\theta = \{a, b\}$

$$f_i(\theta) = \beta_{ij} g_j(\theta) + r_i(\theta) \ .$$

An appropriate (Bayes linear or probabilistic) prior specification must be made for β and $r(.)$.

Working within this framework, we represent all of the following kinds of uncertainty [Vernon et al., 2010]:

1. **Simulator uncertainty:** $f(\theta)$ represents our simulator, which is assumed to be a deterministic function of its inputs. Until $f(.)$ has been run at a particular setting of θ its value at that location is unknown, and so is treated as a random variable within a subjective Bayesian framework. We expect that points which are close together in input space will produce similar function output values- we therefore generally specify that our emulators are made up of a regression component $\beta_{ij} g_j(.)$ and residual component $r_i(.)$, with the regression terms

being used to absorb any global effects and the residual terms capturing any local deviations from these values. We make a prior (probabilistic or second-order) specification for the β and for $r(.)$, and we update these beliefs using runs from the simulator, as described in Section 2.2. Modelling the simulator in this way can produce large savings in computation time, as the emulator is generally much cheaper to run than the simulator which it represents.

2. **Parameter uncertainty:** Uncertainty about the input setting a^* which best reproduces system behaviour can arise in a number of different ways; in some situations, these inputs may be a property of the model only (for example, the resolution of the grid used to solve a differential equation), whereas in others, the model inputs may correspond to idealised representations of measured quantities (for example, the ground properties in Huntley and Goldstein [2016]), properties of processes which are believed to exist but which cannot be directly measured (for example, the effect of dark matter on the galaxy formation simulator in Vernon et al. [2010]), or real-world quantities which we are just uncertain about (because they have been measured with error, or not measured at all). In all of these cases, our treatment is the same, we specify our beliefs through a probability distribution and propagate this uncertainty through the emulator as described in Section 2.4.2.

3. **Observational error:** The observational error (or noise) term $\epsilon(\theta)$ captures imperfections in the observations on the system; the noise can be either correlated between inputs or uncorrelated, depending on prior beliefs about the data-gathering process. Perhaps the simplest and most common specification here is that ϵ has mean zero and is uncorrelated throughout the input space; this is generally used to represent direct but imperfect measurements of particular aspects of the system. However, for more complex problems, data may be obtained indirectly by propagating observed quantities through additional processes to obtain quantities that can be compared with model predictions-this can cause additional complexities in the error structure, as in the Galaxy formation example considered by Vernon et al. [2010].

4. **Simulator discrepancy:** Even if we had access to enough computer power to evaluate the simulator at every input setting of interest, unless the system under study is extremely simple (e.g. a pendulum in a controlled environment), the simulator will still give an incomplete representation of the real world. For instance, the Gaussian plume description of the transport of particulate matter discussed in Section 1.1.3 neglects some of the physics represented in the underlying equation, which in turn neglects some of the turbulent effects that would be present in a more complex description of the behaviour of the atmosphere. This is handled in (2.3.17) through $\delta(.)$, which encodes our beliefs about the discrepancy between the simulator and the real system. This term may contain both systematic and non-systematic components.

After making a prior specification for all components described above, we begin by updating the emulator for the simulator as described in Section 2.2; then we propagate the uncertainty on the parameters a^* through this emulator, and use the resulting simulator predictions in conjunction with data observed on the system to estimate the discrepancy $\delta(.)$ and to further update $\hat{f}(.)$. Details of this calculation when using Bayes linear assumptions are given in Section 2.4.2. If required, we can also use the data observed on the system to learn about settings of a^* which give an acceptable match to the system; we can do this either through calibration (Section 2.4.3) or history matching (Section 2.4.4).

2.3.2 Inference for integrals and derivatives

Gaussian processes and second-order emulators have the extremely useful property that expectations and covariances of derivatives and integrals of the random function can be computed directly from the mean and covariance functions of the process itself [Yaglom, 1986]. This means that such processes can be useful tools for numerical integration, and for the approximation of solutions of differential equations.

Derivatives: If we make a prior specification for an unknown function, then, implicitly, we are also making a prior specification for all of its derivatives. If we specify

$\mathrm{E}\left[f_i\left(\theta\right)\right]$ and $\mathrm{Cov}\left[f_i\left(\theta\right), f_j\left(\theta'\right)\right]$ for a function $f\left(.\right)$, then we also have that

$$\mathrm{E}\left[\frac{\partial}{\partial\theta_k}\big(f_i\left(\theta\right)\big)\right] = \frac{\partial}{\partial\theta_k}\Big(\mathrm{E}\left[f_i\left(\theta\right)\right]\Big)$$

we can also determine covariances between derivatives evaluated at different input settings as

$$\mathrm{Cov}\left[\frac{\partial}{\partial\theta_k}\big(f_i\left(\theta\right)\big), \frac{\partial}{\partial\theta'_l}\big(f_j\left(\theta'\right)\big)\right] = \frac{\partial}{\partial\theta_k}\frac{\partial}{\partial\theta'_l}\Big(\mathrm{Cov}\left[f_i\left(\theta\right), f_j\left(\theta'\right)\right]\Big)$$

and covariances between the original function and its derivatives are

$$\mathrm{Cov}\left[f_i\left(\theta\right), \frac{\partial}{\partial\theta'_l}\big(f_j\left(\theta'\right)\big)\right] = \frac{\partial}{\partial\theta'_l}\Big(\mathrm{Cov}\left[f_i\left(\theta\right), f_j\left(\theta'\right)\right]\Big) \ .$$

Expectations and covariances for higher-order derivatives can be computed by taking further derivatives of the moments in the same way.

This property has a number of potential uses:

- where it is difficult to algebraically compute the derivative of a complex function, we can fit an emulator as an approximation, and then use the emulator to predict the derivative at any input setting;
- it is possible to obtain derivative information from some computer simulators almost 'for nothing', since such models generally solve systems of differential equations, and so pass around derivative information which can easily be exploited. These derivative observations can be used in conjunction with the above relationships in order to improve the quality of the fit; such an analysis is carried out by Killeya [2004] for a simple, compartmental ecosystem model;
- linear differential equation problems relate different derivative states of the solution to each other, and to a set of parameters and forcings; making a prior specification for the solution function and computing the corresponding moments for the operator-transformed function allows us to infer the solution from the forcing, or the forcing from the solution. Several such problems are considered by Graepel [2003].

Care should be taken when using an emulator fitted to a function to reason about its derivatives; the fact that a particular model performs well in validation checks

against samples of the function does not mean that it would pass the same checks against observations of the function's derivatives. Before we can make predictions of derivatives with confidence, we must confirm that we actually believe the smoothness properties of the mean and covariance specifications that we have used.

Integrals: If we perform the integral of a Gaussian process or second-order emulator against another, known function, the resulting stochastic process is of the same type as the original (see O'Hagan [1991] for a demonstration of the Gaussian case); consider the case where we integrate the product of the random function $f(.)$ and a known function $g(.)$ with respect to a single, scalar input a (where the full input set is $\theta = \{a, b\}$) between known limits

$$\bar{f}_i(l_a, u_a, b) = \int_{l_a}^{u_a} f_i(a, b)\, g(a) da \ .$$

For both second-order and probabilistic specifications, we can compute the mean and covariance functions of $\bar{f}(.)$ directly from those of the original process

$$\begin{aligned} \mathrm{E}\left[\bar{f}_i(l_a, u_a, b)\right] &= \int_{l_a}^{u_a} \mathrm{E}\left[f_i(a, b)\right] g(a) da \\ \mathrm{Cov}\left[\bar{f}_i(l_a, u_a, b), \bar{f}_j(l'_a, u'_a, b')\right] &= \int_{l_a}^{u_a} \int_{l'_a}^{u'_a} g(a)\mathrm{Cov}\left[f_i(a, b), f_i(a', b')\right] g(a') da da' \ . \end{aligned}$$

The ease with which we can carry out these calculations depends on the mean and covariance functions, and on the function $g(.)$. It is similarly easy to compute the covariance between the integral with respect to $g(.)$ and the original functon

$$\mathrm{Cov}\left[\bar{f}_i(l_a, u_a, b), f_j(\theta')\right] = \int_{l_a}^{u_a} g(a)\mathrm{Cov}\left[f_i(a, b), f_i(\theta')\right] da \ .$$

These properties extend in an obvious manner to integrals in multiple input parameters. In the case where the covariance function is separable in all parameters and there is no dependence between the limits of the integrals for different inputs, then the covariance of the integral can be computed as a product of terms for the individual inputs; introducing dependency between the integral limits, or using non-separable covariance functions can introduce additional difficulties into the above calculations, but these are not necessarily insurmountable.

These properties have received much attention in recent years as a means of performing integrals numerically; Diaconis [1986] discusses the general case in which calculations of the above form are used as an alternative to numerical integration techniques, and O'Hagan [1991] and Rasmussen and Ghahramani [2003] both consider the use of Gaussian processes as an alternative to traditional stochastic Monte-Carlo methods. Indeed, O'Hagan [1987] goes a step further and argues that traditional Monte-Carlo methods are not Bayesian, since their justification relies on the use of limit theorems and they throw away information about the shape of the distribution under study, and proposes that approximation using random functions is superior. As pointed out by Diaconis [1986], while we may know a great deal about the properties of a function (particularly in simple cases), we do not know its value at a given point until we have actually evaluated it there; in numerical problems, therefore, just as in any other problem, we should adopt a Bayesian approach, fitting a model to the function and then using the model to reason about the integral. Performing integrals in this way conveys a number of advantages over traditional numerical integration methods, in which we evaluate the function at a number of specially-chosen knots and then compute a weighted sum. The Bayesian approach:

- generates an estimate of the integral, but also generates our uncertainty about this estimate; if we are happy that the resulting level of uncertainty is sufficiently small for our purposes, then we may proceed, but if our uncertainty is still too great, then we may repeat the calculation, having made more evaluations of the original function, in the hope of reducing it. If we were happy with the fit to the original function, then we can be assured that this uncertainty has been propagated through coherently to our uncertainty about the integral. Under traditional quadrature approaches, it is very hard to know how good the approximation of the integral is.

- scales much better with dimension; while traditional numerical approaches require evaluations on a specific grid or mesh, under the Bayesian approach, evaluations can be made anywhere in the space in such a way as to provide the best information about the parameters and residual process under a particular specification.

- provides us with an opportunity to build in beliefs that we hold about function properties; for example, we may know that a particular function is twice differentiable, and we can choose mean and covariance functions which also have this property.

Recently, this approach to numerical integration has been used to propagate uncertainty through ordinary differential equation models; for example, if we fit a Gaussian process to the right-hand side of a first order equation, then we may integrate this process over a short time step, before plugging the resulting distribution for the new state back into the RHS again, giving a distribution for the trajectory of the solution, rather than a point estimate. For details of this approach, see, for example, Chkrebtii et al. [2016].

We perform the calculations necessary for learning about the integral of a random function in Section 2.4.1; we will use this approach to evaluate difficult numerical integrals that we encounter when solving complex sequential design problems in chapter 3.

2.4 Performing calculations on random functions

In this Section, we explicitly carry out some of the calculations detailed in Section 2.3 for second-order emulators: in Section 2.4.1, we transfer our adjusted beliefs about a random function to our corresponding beliefs about the function's integral by directly integrating the moments, and in Section 2.4.2, we perform the calculations necessary to propagate input uncertainty through an emulator for a simulator, and adjust the resulting moments and discrepancy by the data observed on the system. Then in Section 2.4.3, we consider using the system data to learn about the best input setting, and in Section 2.4.4, we consider using history-matching to carry out the same task.

2.4.1 Bayesian quadrature

Suppose that we have constructed a second-order emulator for a function in the usual way (as detailed in Section 2.2.2), starting with prior moment specifications

for the basis coefficients and residual process, and then adjusting these moments using a number of runs on the function itself. We split the input parameters into two subsets, $\theta = \{a, b\}$, and we wish to compute our corresponding moment specification for $\bar{f}(b)$, the expectation of $f(\theta)$ with respect to $p(a)$

$$\bar{f}_i(b) = \int_{\mathcal{A}} f_i(a,b)\, p(a)\, da \ .$$

As discussed in Section 2.3.2, this is another second-order process, and we can obtain its moments by integrating the mean and covariance functions directly. Integrating the expression (2.2.12) for the adjusted expectation at a new point, we obtain

$$\begin{aligned} \mathrm{E}_F\left[\bar{f}_i(b)\right] &= \int_{\mathcal{A}} \mathrm{E}_F\left[f_i(a,b)\right] p(a)\, da \qquad (2.4.18) \\ &= \int_{\mathcal{A}} \left[\mathrm{E}_F\left[\beta_{ik}\right] g_k(a,b) + \mathrm{E}_F\left[r_i(a,b)\right]\right] p(a)\, da \\ &= \mathrm{E}_F\left[\beta_{ik}\right] \bar{g}_k(b) + \mathrm{E}_F\left[\bar{r}_i(b)\right] \qquad (2.4.19) \end{aligned}$$

where

$$\bar{g}_k(b) = \int_{\mathcal{A}} g_k(a,b)\, p(a)\, da$$

and $\bar{r}_i(b)$ is the expectation of the residual process over a, whose adjusted moments can be computed as

$$\mathrm{E}_F\left[\bar{r}_i(b)\right] = \mathrm{Cov}\left[\bar{r}_i(b), r_k(\theta_l)\right] \mathrm{Var}\left[F\right]^{-1}_{klpq} \left[F_{pq} - \mathrm{E}\left[F_{pq}\right]\right]$$

with

$$\mathrm{Cov}\left[\bar{r}_i(b), r_k(\theta_l)\right] = \int_{\mathcal{A}} k_{ik}(a,b,\theta_l)\, p(a)\, da$$

where $k(.,.)$ is the covariance function for the residual process. The covariance function for the integral can be computed by starting from (2.2.13) and integrating in both arguments

$$\begin{aligned} \mathrm{Cov}_F\left[\bar{f}_i(b), \bar{f}_j(b')\right] &= \bar{g}_k(b)\, \mathrm{Cov}_F\left[\beta_{ik}, \beta_{jl}\right] \bar{g}_l(b') + \mathrm{Cov}_F\left[\bar{r}_i(b), \bar{r}_j(b')\right] \qquad (2.4.20) \\ &\quad + \bar{g}_k(b)\, \mathrm{Cov}_F\left[\beta_{ik}, \bar{r}_j(b')\right] + \mathrm{Cov}_F\left[\bar{r}_i(b), \beta_{jl}\right] \bar{g}_l(b') \\ &\qquad (2.4.21) \end{aligned}$$

where

$$\mathrm{Cov}_F\left[\beta_{ik}, \bar{r}_j(b')\right] = -\mathrm{Cov}\left[\beta_{ik}, \beta_{pv}\right] G_{vq} \mathrm{Var}\left[F\right]^{-1}_{pqrs} \mathrm{Cov}\left[r_r(\theta_s), \bar{r}_j(b')\right]$$

with a similar expression for the final term, and

$$\begin{aligned}\mathrm{Cov}_F\left[\bar{r}_i\left(b\right),\bar{r}_j\left(b'\right)\right] =& \mathrm{Cov}\left[\bar{r}_i\left(b\right),\bar{r}_j\left(b'\right)\right] \\ &- \mathrm{Cov}\left[\bar{r}_i\left(b\right),r_k\left(\theta_l\right)\right]\mathrm{Var}\left[F\right]^{-1}_{klpq}\mathrm{Cov}\left[r_p\left(\theta_q\right),\bar{r}_j\left(b'\right)\right]\end{aligned}$$

with

$$\mathrm{Cov}\left[\bar{r}_i\left(b\right),\bar{r}_j\left(b'\right)\right] = \int_{\mathcal{A}}\int_{\mathcal{A}} k_{ij}\left(a,b,a',b'\right)p\left(a\right)p\left(a'\right)dada' \ .$$

In this instance, if we were to start with a Gaussian process assumption, the calculations for deriving the expectation and variance of the integral would be exactly the same (O'Hagan [1991] considers the univariate case in detail), the only difference being that we would obtain another Gaussian process for $\bar{f}\left(b\right)$, with $\mathrm{E}_F\left[\bar{f}\left(b\right)\right]$ as its mean function and $\mathrm{Cov}_F\left[\bar{f}\left(b\right),\bar{f}\left(b'\right)\right]$ as its covariance function.

2.4.2 Input uncertainty propagation and uncertainty analysis

In Section 2.3.1, we outlined the standard relationship assumed between a simulator and the system that it represents when carrying out a Bayesian uncertainty analysis; now, for a Bayes linear specification, we compute our moments of $\hat{f}\left(b\right) = f\left(a^*,b\right)$ by propagating uncertainty about a^* through the emulator, and then adjust these moments jointly with the discrepancy using data observed on the system.

Propagating input uncertainty As in equation 2.3.17, we have that $y_i\left(b\right) = \hat{f}_i\left(b\right) + \delta_i\left(b\right)$, and so to compute the moments of $y\left(.\right)$, we must first compute the moments of $\hat{f}\left(.\right)$ for any input setting. We do this as follows, where all outer expectations and covariances are taken with respect to our beliefs $p\left(a^*\right)$ about the best input setting

$$\begin{aligned}\mathrm{E}\left[\hat{f}_i\left(b\right)\right] &= \mathrm{E}\left[\mathrm{E}_F\left[f_i\left(a^*,b\right)\right]\right] \\ \mathrm{Cov}\left[\hat{f}_i\left(b\right),\hat{f}_j\left(b'\right)\right] &= \mathrm{E}\left[\mathrm{Cov}_F\left[f_i\left(a^*,b\right),f_j\left(a^*,b'\right)\right]\right] \\ &\quad + \mathrm{Cov}\left[\mathrm{E}_F\left[f_i\left(a^*,b\right)|a^*\right],\mathrm{E}_F\left[f_j\left(a^*,b'\right)\right]\right] \ .\end{aligned}$$

The expectation $\mathrm{E}_F\left[\hat{f}_i(b)\right]$ is equivalent to the expectation $\mathrm{E}_F\left[\bar{f}_i(b)\right]$ computed in (2.4.19). To compute the covariance, we begin by expanding the second term

$$\begin{aligned}\mathrm{Cov}\left[\hat{f}_i(b), \hat{f}_j(b')\right] &= \mathrm{E}\left[\mathrm{Cov}_F\left[f_i(a^*, b), f_j(a^*, b')\right]\right] \\ &\quad + \mathrm{E}\left[\mathrm{E}_F\left[f_i(a^*, b)\right]\mathrm{E}_F\left[f_j(a^*, b')\right]\right] \\ &\quad - \mathrm{E}_F\left[\hat{f}_i(b)\right]\mathrm{E}_F\left[\hat{f}_j(b')\right] .\end{aligned}$$

The expectation of the covariance is computed as

$$\begin{aligned}\mathrm{E}\left[\mathrm{Cov}_F\left[f_i(a^*, b), f_j(a^*, b')\right]\right] = &\ \mathrm{Cov}_F\left[\beta_{ik}, \beta_{jl}\right] h_{k,l}(b, b') \\ &+ u_{i,j}(b, b') - \mathrm{Var}\left[F\right]^{-1}_{klpq} w_{ikl,jpq}(b, b') \\ &- \mathrm{Cov}\left[\beta_{ik}, \beta_{pv}\right] G_{vq} \mathrm{Var}\left[F\right]^{-1}_{pqrs} v_{k,rsj}(b, b') \\ &- \mathrm{Var}\left[F\right]^{-1}_{pqrs} G_{ws} \mathrm{Cov}\left[\beta_{rw}, \beta_{jl}\right] v_{l,pqi}(b', b)\end{aligned}$$

where we have introduced the following functions

$$\begin{aligned}u_{i,j}(b, b') &= \int_{\mathcal{A}} k_{ij}(a^*, b, a^*, b')\, p(a^*)\, da^* \\ h_{k,l}(b, b') &= \int_{\mathcal{A}} g_k(a^*, b)\, g_l(a^*, b')\, p(a^*)\, da^* \\ v_{k,rsj}(b, b') &= \int_{\mathcal{A}} g_k(a^*, b)\, k_{rj}(\theta_s, a^*, b')\, p(a^*)\, da^* \\ w_{ikl,jpq}(b, b') &= \int_{\mathcal{A}} k_{ik}(a^*, b, \theta_l)\, k_{jp}(a^*, b', \theta_q)\, p(a^*)\, da^* .\end{aligned}$$

The second term can then be found as

$$\begin{aligned}\mathrm{E}\left[\mathrm{E}_F\left[f_i(a^*, b)\right]\mathrm{E}_F\left[f_j(a^*, b')\right]\right] = &\ \mathrm{E}_F\left[\beta_{ik}\right]\mathrm{E}_F\left[\beta_{jl}\right] h_{k,l}(b, b') \\ &+ \mathrm{E}_F\left[\beta_{ik}\right] v_{k,pqj}(b, b') W_{pq} \\ &+ W_{pq} v_{l,pqi}(b', b)\, \mathrm{E}_F\left[\beta_{jl}\right] \\ &+ W_{kl} w_{ikl,jrs}(b, b') W_{rs}\end{aligned}$$

where

$$W_{kl} = \mathrm{Var}\left[F\right]^{-1}_{klpq}\left[F_{pq} - \mathrm{E}\left[F_{pq}\right]\right] .$$

Our ability to perform these calculations in closed-form depends on our choice of basis and covariance functions; the calculations are performed for certain combinations of basis and covariance function in appendix B.

System and data moments Having computed the moments of the simulator evaluated at its best input, we can compute the moments of the system, using our specification that the discrepancy is a priori uncorrelated with the simulator and the best input setting

$$\begin{aligned} \mathrm{E}\left[y_i\left(b\right)\right] &= \mathrm{E}\left[\hat{f}_i\left(b\right)\right] + \mathrm{E}\left[\delta_i\left(b\right)\right] \\ \mathrm{Cov}\left[y_i\left(b\right), y_j\left(b'\right)\right] &= \mathrm{Cov}\left[\hat{f}_i\left(b\right), \hat{f}_j\left(b'\right)\right] + \mathrm{Cov}\left[\delta_i\left(b\right), \delta_j\left(b'\right)\right] \ . \end{aligned}$$

For $\mathrm{E}\left[\delta\left(.\right)\right]$ and $\mathrm{Cov}\left[\delta\left(.\right), \delta\left(.'\right)\right]$, we assume the same form for the discrepancy as we did for our original emulator

$$\delta_i\left(b\right) = \beta_{ip}^{(\delta)} g_p^{(\delta)}\left(b\right) + r_i^{(\delta)}\left(b\right) \ .$$

The prior moments of the full discrepancy are then obtained by specifying moments $\mathrm{E}\left[\beta^{(\delta)}\right]$, $\mathrm{Var}\left[\beta^{(\delta)}\right]$ and $\mathrm{Cov}\left[r_i^{(\delta)}\left(.\right), r_j^{(\delta)}\left(.'\right)\right]$ (assuming, as usual, prior independence of the components, mean zero residual) and combining.
The expectations and covariances of the observed data are then

$$\mathrm{E}\left[z_{ij}\right] = \mathrm{E}\left[y_i\left(b_j\right)\right] \tag{2.4.22}$$

$$\mathrm{Cov}\left[z_{ij}, z_{kl}\right] = \mathrm{Cov}\left[y_i\left(b_j\right), y_k\left(b_l\right)\right] + \mathrm{Cov}\left[\epsilon_i\left(b_j\right), \epsilon_k\left(b_l\right)\right] \tag{2.4.23}$$

where we have used our assumption that the measurement error ϵ is uncorrelated with the system y

Adjusting using system data On observing the data, we use this to adjust the moments of the system for new input values; using $\mathrm{Var}\left[z\right]_{ijkl} = \mathrm{Cov}\left[z_{ij}, z_{kl}\right]$ to denote the full variance matrix of the data, and $\mathrm{Var}\left[z\right]^{-1}_{ijkl}$ to denote an element of its inverse, the adjusted moments of the system are

$$\begin{aligned} \mathrm{E}_z\left[y_i\left(b\right)\right] &= \mathrm{E}_z\left[\hat{f}_i\left(b\right)\right] + \mathrm{E}_z\left[\delta_i\left(b\right)\right] \\ \mathrm{Cov}_z\left[y_i\left(b\right), y_j\left(b'\right)\right] &= \mathrm{Cov}_z\left[\hat{f}_i\left(b\right), \hat{f}_j\left(b'\right)\right] + \mathrm{Cov}_z\left[\delta_i\left(b\right), \delta_j\left(b'\right)\right] \\ &\quad + \mathrm{Cov}_z\left[\hat{f}_i\left(b\right), \delta_j\left(b'\right)\right] + \mathrm{Cov}_z\left[\delta_i\left(b\right), \hat{f}_j\left(b'\right)\right] \ . \end{aligned}$$

For the best input simulator value and the discrepancy, we compute the adjustments as

$$\begin{aligned}
\mathrm{E}_z\left[\hat{f}_i(b)\right] &= \mathrm{E}\left[\hat{f}_i(b)\right] + \mathrm{Cov}\left[\hat{f}_i(b), z_{kl}\right] \mathrm{Var}[z]^{-1}_{klpq}\left[z_{pq} - \mathrm{E}[z_{pq}]\right] \\
\mathrm{Cov}_z\left[\hat{f}_i(b), \hat{f}_j(b')\right] &= \mathrm{Cov}\left[\hat{f}_i(b), \hat{f}_j(b')\right] \\
&\quad - \mathrm{Cov}\left[\hat{f}_i(b), z_{kl}\right] \mathrm{Var}[z]^{-1}_{klpq} \mathrm{Cov}\left[z_{pq}, \hat{f}_j(b')\right]
\end{aligned}$$

where

$$\mathrm{Cov}\left[\hat{f}_i(b), z_{kl}\right] = \mathrm{Cov}\left[\hat{f}_i(b), \hat{f}_k(b_l)\right]$$

and

$$\begin{aligned}
\mathrm{E}_z[\delta_i(b)] &= \mathrm{E}_z\left[\beta^{(\delta)}_{ip}\right] g^{(\delta)}_p(b) + \mathrm{E}_z\left[r^{(\delta)}_i(b)\right] \\
\mathrm{Cov}_z[\delta_i(b), \delta_j(b')] &= g^{(\delta)}_p(b)\, \mathrm{Cov}_z\left[\beta^{(\delta)}_{ip}, \beta^{(\delta)}_{jq}\right] g^{(\delta)}_q(b') + \mathrm{Cov}_z\left[r^{(\delta)}_i(b), r^{(\delta)}_j(b')\right] \\
&\quad g^{(\delta)}_p(b)\, \mathrm{Cov}_z\left[\beta^{(\delta)}_{ip}, r^{(\delta)}_j(b')\right] + \mathrm{Cov}_z\left[r^{(\delta)}_i(b), \beta^{(\delta)}_{jq}\right] g^{(\delta)}_q(b')
\end{aligned}$$

where the individual components of this adjustment are computed in the same way as those of the Bayes linear emulator update described in Section 2.2.2, substituting moments of F for moments of z where appropriate. It then only remains to compute the adjusted covariance between the best input simulator and the discrepancy, which is

$$\begin{aligned}
&\mathrm{Cov}_z\left[\hat{f}_i(b), \delta_j(b')\right] = \\
&- \mathrm{Cov}\left[\hat{f}_i(b), z_{kl}\right] \mathrm{Var}[z]^{-1}_{klpq}\left[G^{(\delta)}_{rq} \mathrm{Cov}\left[\beta^{(\delta)}_{pr}, \beta^{(\delta)}_{js}\right] g^{(\delta)}_s(b') + \mathrm{Cov}\left[z_{pq}, r^{(\delta)}_j(b')\right]\right]
\end{aligned}$$

where $G^{(\delta)}_{rq} = g^{(\delta)}_r(b_q)$ is the design matrix for the discrepancy basis functions and

$$\mathrm{Cov}\left[z_{pq}, r^{(\delta)}_j(b')\right] = \mathrm{Cov}\left[r^{(\delta)}_p(b_q), r^{(\delta)}_j(b')\right] .$$

2.4.3 Inferring the inputs: calibration

When using an emulator as a representation for a system, we are often interested in the question of what observations on the system can tell us about the best input parameters to the model: which parts of the input space for a are likely to have produced particular observations? Relatively how likely are different parts of the

input space? Using system data to learn about the parameters in this way is referred to as calibration; Kennedy and O'Hagan [2001] considered the calibration of a Gaussian process model in this way, using a fully Bayesian framework in which the observations define a posterior distribution over the model input space, and future predictions are calibrated by numerically integrating the predictive distribution against this posterior. Here, we consider the Bayes linear calibration methods of Goldstein and Rougier [2006], in which a suitable second-order model specification leads to us being able to derive all of the moments necessary to adjust beliefs about the input parameters using the data.

The moments of the data observed on the system were computed in equations (2.4.22) and (2.4.23); the Bayes linear adjustment equations for inputs a_i^* are now simply

$$\begin{aligned} \mathrm{E}_z\left[a_i^*\right] &= \mathrm{E}\left[a_i^*\right] + \mathrm{Cov}\left[a_i^*, z_{kl}\right] \mathrm{Var}\left[z\right]^{-1}_{klpq} \left[z_{pq} - \mathrm{E}\left[z_{pq}\right]\right] \\ \mathrm{Cov}_z\left[a_i^*, a_j^*\right] &= \mathrm{Var}\left[a_i^*, a_j^*\right] - \mathrm{Cov}\left[a_i^*, z_{kl}\right] \mathrm{Var}\left[z\right]^{-1}_{klpq} \mathrm{Cov}\left[z_{pq}, a_j^*\right] \ . \end{aligned}$$

In order to carry out this adjustment, we need to compute the covariances $\mathrm{Cov}\left[a_i^*, z_{kl}\right]$ as

$$\mathrm{Cov}\left[a_i^*, z_{jk}\right] = \mathrm{E}\left[a_i^* \mathrm{E}_F\left[f_j\left(a^*, b_k\right)\right]\right] - \mathrm{E}\left[a_i^*\right] \mathrm{E}\left[\mathrm{E}_F\left[f_j\left(a^*, b_k\right)\right]\right]$$

where again, the outer expectations are computed with respect to our beliefs $p\left(a^*\right)$ about the best input setting. The first term is then

$$\begin{aligned} \mathrm{E}\left[a_i^* \mathrm{E}_F\left[f_j\left(a^*, b_k\right)\right]\right] = \ & \mathrm{E}_F\left[\beta_{jp}\right] \left[\int_{\mathcal{A}} a_i^* g_p\left(a^*, b_k\right) p\left(a^*\right) da^*\right] \\ & + W_{pq} \left[\int_{\mathcal{A}} a_i^* k_{jp}\left(a^*, b_k, \theta_q\right) p\left(a^*\right) da^*\right] \end{aligned}$$

where W is defined as in Section 2.4.2.

Caution should be exercised when using the Bayes linear approach to learning about the inputs. If the behaviour of the function $f\left(.\right)$ is highly variable across the input space, then it may be the case that there are multiple different parts of the space which will give good matches to the system; if this is the case, then the probabilistic approach of Kennedy and O'Hagan may be preferable, since it has the ability to capture this multi-modality. In other cases, it may be that there is no setting of the

inputs that gives a match to the system data which we would consider acceptable; in this instance, calibration is not an appropriate technique, since it will always result in a set of moments, or a normalised probability distribution. The paper by Brynjarsdottir and O'Hagan [2014] contains a detailed discussion of this point; in this situation, we should explore the input space using history matching, as in Section 2.4.4.

Having used the system data to learn about the best input setting a^*, it is natural to ask about the benefit that we could obtain by re-running the full simulator at this point; obtaining this run and re-adjusting our emulator to include it has the potential to significantly decrease our uncertainty in an important part of the input space. The paper by Goldstein and Rougier [2006] considers the benefit that we might obtain by re-running the simulator at $\mathrm{E}_z[a^*]$ by computing the joint moment specification for a^* and $f(\mathrm{E}_z[a^*], b)$.

2.4.4 Inferring the inputs: history matching

An alternative approach to learning about the input space of the model is known as history matching [Vernon et al., 2010]. Instead of using the observations to perform a Bayes linear adjustment, or to define a full posterior probability distribution, we proceed by means of a discrepancy measure between the system prediction and the data observed on the system itself.

Rather than attempting to integrate out a best setting of the simulator inputs a, we compute moments for the data as a function of these inputs by combining our emulator (after adjustment by F) with our prior beliefs about the discrepancy and about the measurement error structure; we then use these moments to define a measure of our degree of surprise at having seen the data z at a particular setting of the simulator inputs, and we use this measure to discriminate between different values of a. As a function of a, then, our beliefs about the data are

$$\begin{aligned}\mathrm{E}[z_{ij}(a)] &= \mathrm{E}_F[f_i(a, b_j)] + \mathrm{E}[\delta_i(b_j)] \\ \mathrm{Cov}[z_{ij}(a), z_{kl}(a)] &= \mathrm{Cov}_F[f_i(a, b_j), f_k(a, b_l)] + \mathrm{Cov}[\delta_i(b_j), \delta_k(b_l)] \\ &\quad + \mathrm{Cov}[\epsilon_i(b_j), \epsilon_k(b_l)] \ .\end{aligned}$$

We define the implausibility measure $I(a)$ to measure the discrepancy between the data and the simulator predictions for this particular value of a. We choose a definition based on the Mahalanobis distance, though other choices can be made; the paper by Vernon et. al. contains further discussion of this point

$$I(a) = \Big(z_{ij} - \mathrm{E}\left[z_{ij}(a)\right]\Big)\Big[\mathrm{Var}\left[z(a)\right]^{-1}\Big]_{ijkl}\Big(z_{kl} - \mathrm{E}\left[z_{kl}(a)\right]\Big) .$$

Higher values of $I(a)$ indicate a greater discrepancy between the simulator prediction at this setting of its inputs and the system data. We use this measure to define a non-implausible space of inputs by imposing a threshold χ above which we believe that it is very unlikely that the data could have been generated by this a; the non-implausible space $\mathcal{A}^*$ then consists of inputs for which $I(a) < \chi$. For the multivariate measure defined above, a suitable choice for χ could be a 95 or 99 percent quantile of the chi-squared distribution; this effectively corresponds to making a Gaussian specification for the each of the model components. Where univariate measures are to be used, a common choice is $\chi = 3$, owing to the result of Pukelsheim [1994] that 95% of the probability mass lies within three standard deviations of the mean for any continuous, unimodal probability distribution.

History matching is generally performed in waves; we fit an initial emulator to the simulator $f(.)$, and we use this emulator (and prior beliefs about discrepancy and measurement error) to define a measure $I_1(a)$ at the first wave. We then use this implausibility measure to sample non-implausible settings of a, before using these points to fit a second emulator to the simulator defined within the non-implausible space $\mathcal{A}_1^*$ at this first wave. Because this emulator is fitted only in a subset of the full input space, we hope to be able to increase the accuracy of its predictions, which will increase our power to discriminate between good and bad settings of a.

The first- and second-wave emulators are then used to define a space $\mathcal{A}_2^*$ of settings a satisfying $I_1(a) < \chi_1$ and $I_2(a) < \chi_2$; this space then has the property that $\mathcal{A}_2^* \subset \mathcal{A}_1^*$, giving us tighter constraints on the settings of a that we consider could give an acceptable match to the system. We can perform as many waves of history matching as we deem necessary; we would typically stop performing waves either when our description $\mathcal{A}_k^*$ of the space of possible best inputs is sufficiently accurate for our purposes, or when we cease to see any significant reduction in the size of the

non-implausible space between successive waves.

2.5 Simple example

We now illustrate the analyses outlined in this chapter through application to a simple example; we generate 10 samples from the following pair of functions at a Latin hypercube of input locations in $b \in [0,1]$, $a \in [0,1]$

$$\begin{aligned} f_1(a,b) &= \sin(2\pi(b+a)) \\ f_2(a,b) &= \cos(2\pi(b+a)) \ . \end{aligned}$$

If we know of underlying structure in the functions that we are trying to model, then we can increase the power of our model by building this structure into it; in this instance, we use the fact that $f_2(.)$ is offset from $f_1(.)$ by a quarter period when selecting our mean and covariance functions.

When setting up our prior, we assume that we are uncertain about the function's behaviour, but that we wish to build in the quarter-period offset property between the two outputs. First, the regression component; we believe that the functions will exhibit non-polynomial behaviour around a constant level, and so we specify that

$$g_1(a,b) = 1 \ .$$

Our beliefs about this constant level are symmetric about zero, and so set all prior basis parameter expectations to zero ($\mathrm{E}[\beta_{ij}] = 0$), and we specify that

$$\begin{aligned} \mathrm{Var}[\beta_{11}] &= 0.1^2 \quad \mathrm{Cov}[\beta_{11}, \beta_{21}] = 0 \\ \mathrm{Cov}[\beta_{21}, \beta_{11}] &= 0 \quad \mathrm{Var}[\beta_{21}] = 0.1^2 \end{aligned} \ .$$

For the covariance functions, we use the following forms

$$k_{ij}(a,b,a',b') = V_{ij} \exp\left(-\frac{\lambda}{2}\left((b+a+\xi_i) - (b'+a'+\xi_j)\right)^2 \right)$$

where the ξ_i are known offset parameters, with $\xi_1 = 0$ and $\xi_2 = \frac{1}{4}$, imposing the usual squared exponential correlation between values of the same output and an offset squared exponential correlation between different output values. The matrix elements V_{ij} determine the degree of marginal covariance between the appropriately

offset function values; we believe the residual of output 1 to be perfectly correlated with the residual of output 2 at a quarter-period offset, and so we specify that $V_{ij} = 1$ for all $\{i, j\}$.

Update Having completed the prior specification, we perform the calculations in Section 2.2.2 and update the emulator. In order to determine the parameters of the covariance function, we check our fit against a new data set: we generate an additional 10 points from the underlying function, fit the model, and then compute the standardised distance from the model prediction to the observed function value, as described in Section 2.2.3. In this particularly simple example, we simply use these standardised distances to manually tune the correlation parameter λ; we settle on the value $\lambda = \frac{1}{0.3^2}$.

Figure 2.1 shows some plots of the fitted emulator: Figure 2.1(a) shows the adjusted mean $\mathrm{E}_F[f_1(a,b)]$ as a function of b and a, with the locations of the design points used to fit the model shown in black; Figure 2.1(b) shows the corresponding adjusted standard deviation, and Figure 2.1(c) shows both $f_1(.)$ and $f_2(.)$ as a function of $(b + a)$.

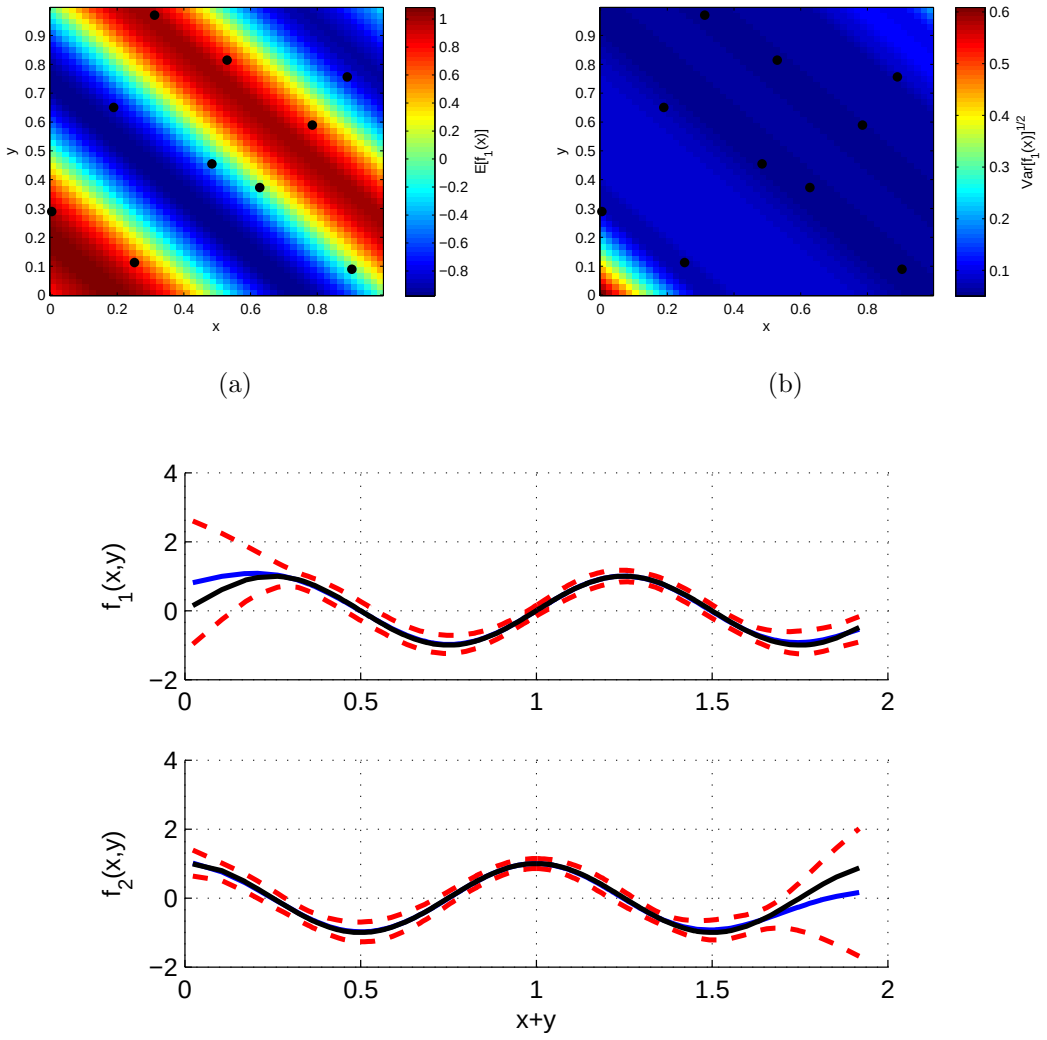
Expectation of the integral We now use our adjusted beliefs to compute our corresponding beliefs about the integrals

$$\bar{f}_i(b) = \int_{\mathcal{Y}} f_i(a,b)\, p(a)\, da \ .$$

We specify that $p(a) = 1$ (a uniform distribution), and compute the moments described in Section 2.3.1. Figure 2.2 shows the moments of $\bar{f}_1(.)$ and $\bar{f}_2(.)$ as a function of b.

Propagation of parameter uncertainty We now use our adjusted beliefs about $f(a,b)$ to compute our corresponding beliefs about $\hat{f}(b) = f(a^*, b)$, given our beliefs $p(a^*)$ about a^*; so we can easily compare the two cases, we also specify that $p(a^*) = 1$, i.e. that the input which gives the best match to the system could lie anywhere within the domain of a.

Figure 2.3 summarizes our beliefs $\mathrm{E}\left[\hat{f}(b)\right]$ and $\mathrm{Var}\left[\hat{f}(b)\right]$ for both outputs; note



(a) (b)

(c)

Figure 2.1: Plots showing the adjusted moments for the example in Section 2.5: Figure 2.1(a) shows the adjusted expectation of $f_1(.)$ as a function of b and a, with the design points used to fit the model shown as black markers; Figure 2.1(b) shows the corresponding adjusted standard deviation; Figure 2.1(c) shows the adjusted moments of both functions as a function of $(b + a)$, with the adjusted expectation shown in blue, two standard-deviation error bars shown in dashed red, and the true underlying function shown in black.

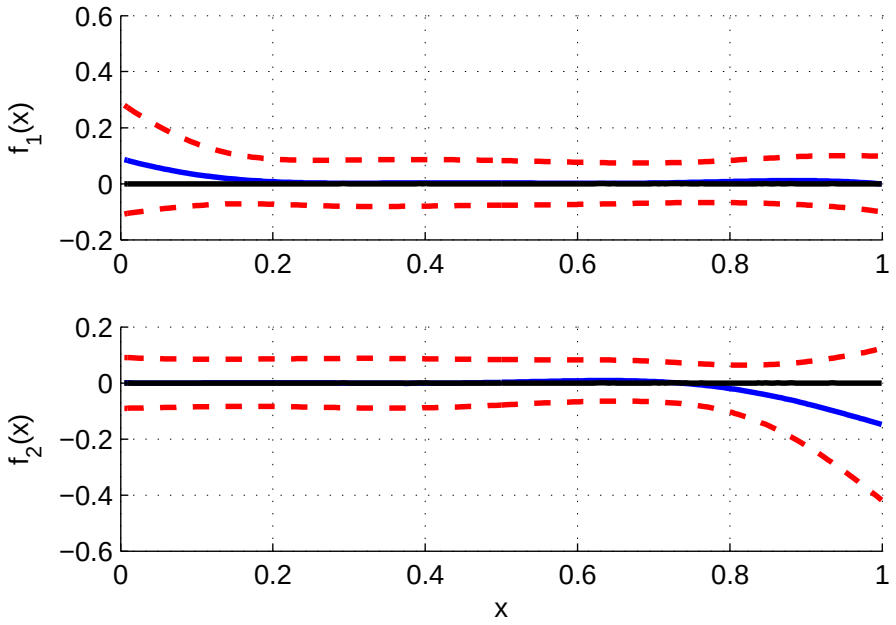


Figure 2.2: Plot of the moments of the integral of the function: the means $\mathrm{E}_F\left[\bar{f}_i(b)\right]$ are shown in blue, and the error bars $\mathrm{E}_F\left[\bar{f}_i(b)\right] \pm 2\mathrm{Var}_F\left[\bar{f}_i(b)\right]^{1/2}$ are shown in dashed red. The true result of the integral is shown in black.

that the mean (solid blue) is the same in both cases as the mean $\mathrm{E}_F\left[\bar{f}(b)\right]$, as outlined in Section 2.4.2. In this case, however, the marginal variance $\mathrm{Var}\left[\hat{f}(b)\right]^{1/2}$ is much larger than $\mathrm{Var}\left[\bar{f}(b)\right]^{1/2}$, since in the former case, the total variance is a sum of both the uncertainty in the mean level due to uncertainty in the setting of a^*, and uncertainty in the emulator for a given setting of a^*.

Learning the uncertain inputs Having computed the moments of the function evaluated at its best input, we make observations of the original function (subject to measurement error) at a particular value of a^*, and use the calculations outlined in Section 2.4.3 to infer this setting. To complete our specification for the observed data, we assume that $\delta(b) = 0$, and so

$$\begin{aligned}\mathrm{E}\left[y_i(b)\right] &= \mathrm{E}\left[\hat{f}_i(b)\right] \\ \mathrm{Cov}\left[y_i(b), y_j(b')\right] &= \mathrm{Cov}\left[\hat{f}_i(b), \hat{f}_j(b')\right]\end{aligned}$$

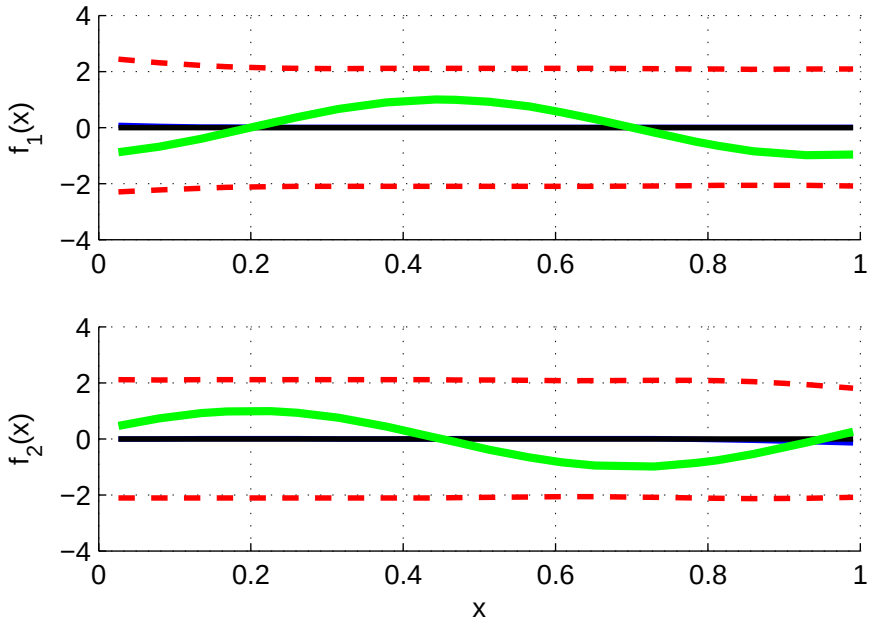


Figure 2.3: Plot of the predictions for the functions $\hat{f}_i(b)$ after the propagation of the uncertainty on a through the function: $\mathrm{E}_F\left[\hat{f}_i(b)\right]$ is shown in blue, and the error bars $\mathrm{E}_F\left[\hat{f}_i(b)\right] \pm 2\mathrm{Var}_F\left[\hat{f}_i(b)\right]$ are shown in dashed red. The green lines show the actual system function $y(b)$ which is used to adjust beliefs about the uncertain inputs a^*. The expectation of the true function is shown in black; the blue line ($\mathrm{E}_F\left[\hat{f}_i(b)\right]$) is almost entirely obscured by this black line.

so that at known observation locations $\{b_k\}$, we have that

$$\begin{aligned} \mathrm{E}\left[z_{ik}\right] &= \mathrm{E}\left[\hat{f}_i\left(b_k\right)\right] \\ \mathrm{Cov}\left[z_{ik}, z_{jl}\right] &= \mathrm{Cov}\left[\hat{f}_i\left(b_k\right), \hat{f}_j\left(b_l\right)\right] + \mathrm{Cov}\left[\epsilon_{ik}, \epsilon_{jl}\right] \end{aligned}$$

where the errors are assumed to be uncorrelated between observations, so $\mathrm{Cov}\left[\epsilon_{ik}, \epsilon_{jl}\right] = (10^{-3})$ only if $i = j$ and $k = l$.

We collect 20 observations at different locations on each of the two outputs; these observations are made at $a^* = 0.8$, and are shown in green in Figure 2.3. The quantities detailed in Section 2.4.3 are then computed, and prior moments $\mathrm{E}\left[a^*\right] = 0.5$ and $\mathrm{Var}\left[a^*\right] = \frac{1}{12} \simeq (0.2887)^2$ (taken from the uniform distribution) are adjusted to

$$\mathrm{E}_z\left[a^*\right] = 0.7961 \qquad \mathrm{Var}_z\left[a^*\right] = (0.1306)^2 \ .$$

Calibrating in this way has resulted in a posterior mean which is very close to the true value at which we generated the data and a reasonably large reduction from the prior standard deviation. Figure 2.4 shows the data used for the adjustment (in green) alongside the moments $\mathrm{E}\left[\hat{f}\left(b\right)\right]$, $\mathrm{Var}\left[\hat{f}\left(b\right)\right]$ re-computed using the uniform distribution parametrized by the adjusted moments $\mathrm{E}_z\left[a^*\right]$ and $\mathrm{Var}_z\left[a^*\right]$.

2.6 Example: Ocean simulator

We now carry out a Bayesian uncertainty analysis for a more complex system; we consider a problem of the type discussed in Section 1.3, where we must link together the output from an ocean simulator with observations on the real ocean in order to generate an uncertainty specification for ocean characteristics that we might observe in the future.

2.6.1 Simulator

The simulator for this system is the third-generation MIKE model for coastal wave behaviour, developed by the Danish Hydrological Institute (DHI). The wave action density spectrum, $N\left(x, y, \phi, \sigma, t\right)$ (which is a function of location (x, y), time t,

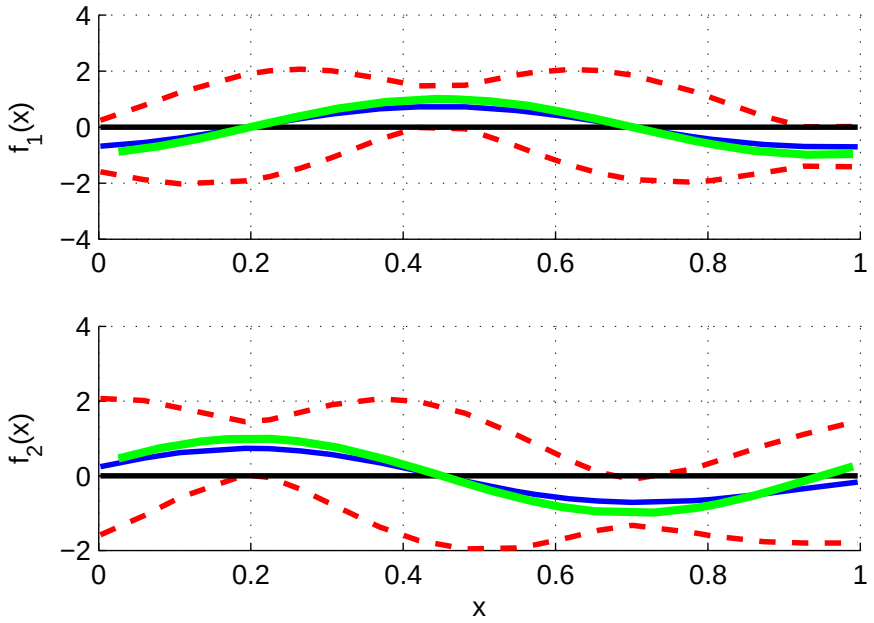


Figure 2.4: Observed data z (green) plotted alongside the moments $\mathrm{E}\left[\hat{f}(b)\right] \pm 3\mathrm{Var}\left[\hat{f}(b)\right]^{1/2}$ (solid blue and dashed red) obtained by re-computing these moments using $\mathrm{E}_z[a^*]$, $\mathrm{Var}_z[a^*]$; we see that our beliefs about the $\hat{f}(.)$ having learned about the input setting a^* correspond much more closely to the underlying 'true' function.

wave direction ϕ and wave period σ) is assumed to evolve in time according to the following wave action conservation equation [Sørensen et al., 2004]

$$\frac{\partial N}{\partial t} + \nabla.(cN) = \frac{s}{\sigma} \; . \tag{2.6.24}$$

Here, $c\,(x, y, \phi, \sigma, t) = (c_x, c_y, c_\phi, c_\sigma)^{\mathrm{T}}$ is the four-dimensional characteristic propagation velocity of a wave-group, and $s\,(x, y, \phi, \sigma, t)$ is an energy source term. In order to make this a well-posed problem, the initial and boundary behaviour of $N\,(.)$ must also be specified. The characteristic propagation velocities can be computed for any input point using a set of kinematic relationships.

The energy source term in this equation is sub-divided into several different components, representing the different sources of energy transfer to the system:

- $s_{in}\,(.)$ represents the energy transferred from the wind;
- $s_{nl}\,(.)$ represents the energy transfer from non-linear interactions between waves;
- $s_{ds}\,(.)$ represents the dissipation of energy due to the white-capping of waves;
- $s_{bot}\,(.)$ represents the dissipation of energy due to bottom friction;
- $s_{surf}\,(.)$ represents the dissipation of energy due to wave-breaking.

Further sets of assumptions are made in order to estimate these quantites from available atmospheric information.

The equation (2.6.24) cannot be solved directly, and so the simulator must be constructed using a numerical solver; Sørensen et al. [2004] use a cell-centred finite-volume method, which exploits the fact that (2.6.24) has the form of a conservation law. A polygonal mesh (tetrahedral or quadrilateral) is imposed on the domain, and the action density spectrum $N\,(.)$ is approximated on each mesh element by a constant (assumed to be located at the centre of the element for the purposes of the solver). Algebraic relationships are then specified between the discrete spectral states at one time-step and the next by integrating (2.6.24) over each mesh element, applying Stokes' theorem and, where necessary, approximating the integrals of the characteristic wave speeds and forcings over the cell boundaries.

Once (2.6.24) has been solved numerically, the resulting wave action density values must be post-processed in order to obtain predictions for the storm characteristics of interest within different sea-states (see Section 1.3). This post processing involves finding moments of the wave spectra for different locations and times. In the case of storm-peak significant wave height, we must also find the maximum wave height within a given storm (which is defined to be an unbroken sequence of sea-states during which wave heights exceed a given threshold- see Section 1.3.1). The outputs that we emulate are outlined in Section 2.6.2.

2.6.2 Emulator

The wave simulator that we use has 5 outputs:

- $f_1(\theta) = \mathrm{Hmp}_{\mathrm{F78}}(\theta)$: Storm-peak significant wave height;
- $f_2(\theta) = \mathrm{Hm0}_{\mathrm{p_{eq}}}(\theta)$: Most probable wave height;
- $f_3(\theta) = \mathrm{Tpwmean}(\theta)$: Peak wave period;
- $f_4(\theta) = \mathrm{T02wmean}(\theta)$: Mean wave period;
- $f_5(\theta) = \log_{10}\left(\mathrm{sigma}_{\mathrm{eq}}(\theta)\right)$: log of the storm duration.

We fit an emulator as a function of inputs $\theta = \{a, b\}$; the b are storm covariates that we will know (or can compute from atmospheric information), and the a are parameters which we must specify in order to be able to run the simulator, but which are not measured on the system. Table 2.1 describes each of the storm covariates used to fit the model, and table 2.2 describes each of the simulator-specific inputs. We begin our model fit by specifying the basis functions and the covariance structure of the residual process. For the vector of basis functions, we choose

$$g(\theta) = \Big(1, I_5(b_1), I_{12}(b_1), I_{13}(b_1), I_{22}(b_1), b_2, b_2^2, b_3^3, \cos(b_3), \sin(b_3), \ldots \\ \cos(b_4), \sin(b_4), a_1, \ldots, a_5, a_1^2, \ldots, a_5^2\Big)^{\mathrm{T}} .$$

That is, we use an overall intercept, an indicator function for each platform except platform 2, first order Fourier terms for the wind direction, polynomial terms up to

Name	Range	Description
$b_1 =$ Platform	{2,5,12,13,22}	Platform (index within full set)
$b_2 =$ WSmax	[1.9,32.4]	Maximum storm wind speed (m/s)
$b_3 =$ WDmax	[0,360]	Direction of max. storm wind (meteorological degrees)
$b_4 =$ Season	[0,365]	Storm season (days)

Table 2.1: Elements of b

3^{rd} order in the wind speed and first- and second-order terms for each of the tuning inputs. The covariance of the residual is assumed to have the simple form discussed in Section 2.2.3

$$\text{Cov}\left[r_i\left(\theta\right), r_j\left(\theta'\right)\right] = V_{ij} c\left(\theta, \theta'\right)$$

where the correlation function is a squared exponential

$$\begin{aligned} c\left(\theta, \theta'\right) &= c_a\left(a, a'\right) c_b\left(b, b'\right) \\ &= \exp\Big[\sum_{k=1}^{5} \frac{\lambda_{a_k}}{2}(a_k - a_k')^2\Big] \\ &\quad \times \exp\Big[\frac{\lambda_{b_2}}{2}(b_2 - b_2')^2 + \frac{\lambda_{b_3}}{2}\sin\big(\frac{b_3 - b_3}{2}\big)^2 + \frac{\lambda_{b_4}}{2}\sin\big(\frac{b_4 - b_4}{2}\big)^2\Big] \ . \end{aligned}$$

This choice of covariance is separable in inputs and outputs, and in all input variables. Using the separability in input and output variables, we can write the covariance of the residual components $R_{ik} = r_i\left(\theta_k\right)$ in the following Kronecker product form

$$\text{Var}\left[\text{Vec}\left(R\right)\right] = \Sigma_\psi \otimes V$$

where $\text{Vec}\left(R\right)$ is the vector obtained by unwrapping the matrix R. This allows us access to the fast inversion procedure outlined at 2.2.3, since we can invert the full residual covariance by inverting each of the above terms separately

$$\text{Var}\left[\text{Vec}\left(R\right)\right]^{-1} = \Sigma_\psi^{-1} \otimes V^{-1} \ .$$

Fitting Having selected the basis and covariance functions, we proceed to fit the emulator. We begin by dividing our data into three; we randomly choose 15000

Name	Range	Description
$a_1 = \mathrm{C}_{\mathrm{diss}}$	[1.7,2.5]	Whitecapping induced dissipation
$a_2 = \mathrm{D}_{\mathrm{diss}}$	[0.3,0.8]	Whitecapping induced dissipation
$a_3 = \mathrm{K}_{\mathrm{n}}$	[1,5]	Roughness size (metres)
$a_4 = \%_{\mathrm{current}}$	[0,3]	Surface current velocity (as a percentage of 10m wind speed)
$a_5 = \mathrm{Triad}$	[0,1]	Triad interactions (on/off)

Table 2.2: Elements of a

points for an initial linear regression, in order to fix our prior specification for the regression weights, and we set aside 500 points which will be used to check our model predictions at each stage. We then randomly choose a set of 3000 points which will be used to fit the emulator.

First, we determine our prior moments for the regression parameters by carrying out a linear regression (in the absence of the residual term) using the 15000-point dataset. The moments $\mathrm{E}\left[\beta_{ip}\right] g_p\left(\theta_k\right)$ that we obtain for each of the 5 outputs are plotted in Figure 2.5 against the true simulator values for the validation set of 500 points; we see that in this problem, the regression surface alone does a good job of capturing the structure in the response.

We then use this initial regression surface to estimate the marginal residual covariance structure across the different outputs. We compute the fitted values $\hat{F}_{ik} = \mathrm{E}\left[\beta_{ip}\right] G_{pk}$ for this regression surface, and compute the residuals $\hat{R}_{ik} = F_{ik} - \hat{F}_{ik}$; we then fix the marginal covariance matrix of the residual to $V = \hat{V}$, where

$$\hat{V}_{ij} = s_i s_j \mathrm{Corr}\left[\hat{R}_i, \hat{R}_j\right]$$

where the empirical marginal standard deviations are

$$s = (0.5785, 1.0396, 0.7380, 0.4152, 0.1794)^{\mathrm{T}}$$

and the empirical residual correlation matrix is

$$\text{Corr}\left[\hat{R}_i, \hat{R}_j\right] = \begin{pmatrix} 1.0000 & 0.9444 & 0.7592 & 0.9179 & -0.0653 \\ 0.9444 & 1.0000 & 0.7379 & 0.8926 & 0.2609 \\ 0.7592 & 0.7379 & 1.0000 & 0.8862 & -0.0080 \\ 0.9179 & 0.8926 & 0.8862 & 1.0000 & 0.0052 \\ -0.0653 & 0.2609 & -0.0080 & 0.0052 & 1.0000 \end{pmatrix}.$$

Outputs $f_1(.)$ to $f_4(.)$ are all quantities that we would expect to be high for the most severe storms, and so we would expect such a high degree of correlation between them.

Having fixed the regression prior and the inter-output covariance, it only remains for us to fix the correlation lengths of the residual process; this is done using a leave-one-out cross validation procedure. Under this procedure, each point is left out of the fitting data set in turn, and the fit to the remainder of the data is used to predict the value of the excluded point; the quality of the fit for a particular setting of the correlation lengths is then evaluated using the sum of the log-Gaussian likelihoods for each of the predictions. To speed up this procedure, we treat the regression parameters as fixed for the purposes of the cross-validation and determine the correlation lengths through fitting to the residuals from the mean regression surface. This allows us to use the algebraic expressions for the predictive means and variances under the leave-one-out procedure presented in Rasmussen and Williams [2006] (Chapter 5), meaning that we must only invert the data covariance matrix once in order to be able to make predictions for each individual point given the rest. A Latin hypercube of 1500 different correlation settings is searched, the correlation parameters are fixed at the setting which minimises the likelihood criterion over this collection, and the regression and residual surfaces are jointly updated at this setting, as described in Section 2.2.2.

Figure 2.6 contains plots of the fitted emulator. Figure 2.6(a) shows predictions from the emulator (vertical axis) against the true simulator output value for each of the 5 outputs for the set of 500 validation points set aside earlier. Figure 2.6(b) shows the corresponding standardised residuals against the true simulator output value. In all windows, the colour of the point corresponds to the platform it was generated

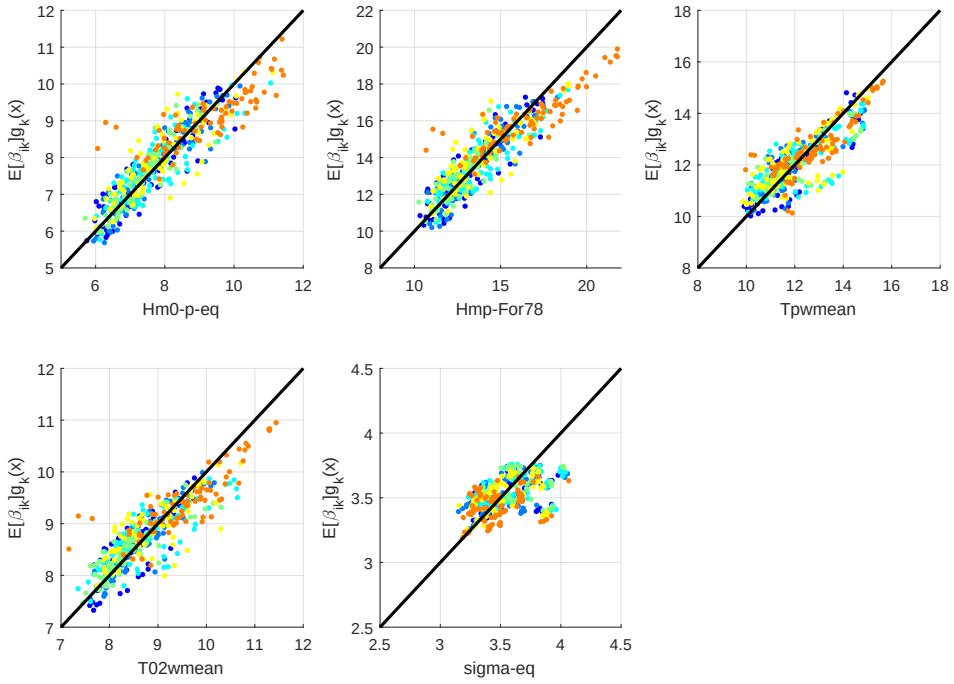
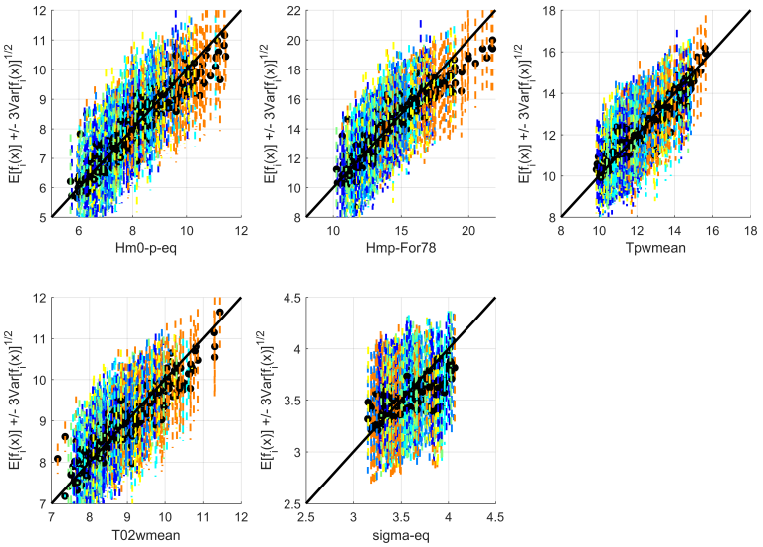


Figure 2.5: Plot of the initial regression fit to the ocean simulator data (Section 2.6.2): in each window, the true simulator output F_{ik} is on the horizontal axis, and the mean prediction $\mathrm{E}\left[\beta_{ip} g_p\left(\theta_k\right)\right]$ from the regression model is on the vertical axis. The points are coloured according to the platforms to which they correspond.

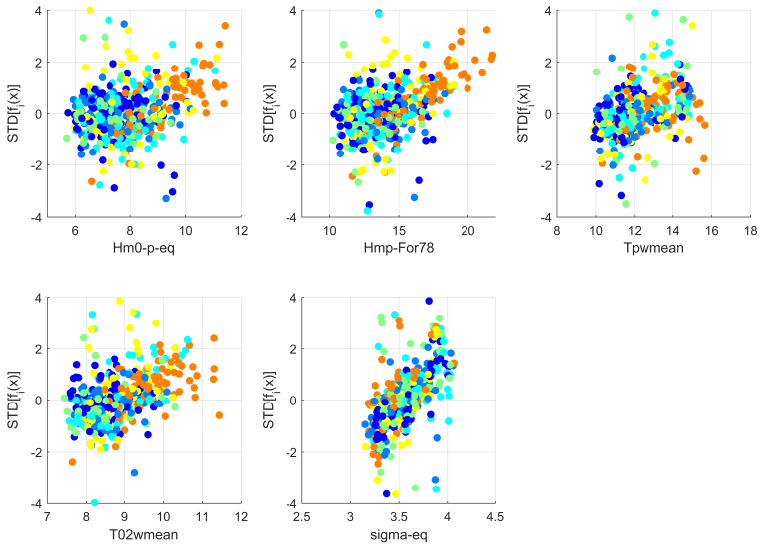
at. Overall, this relatively simple emulator is able to capture a large amount of the structure in the data.

2.6.3 System model

Having fitted an emulator to the ocean simulator, we use a set of observations of real ocean properties to link the simulator to the system, as described in Section 2.4.2. A total of 4182 data points are available from the platforms for which our emulator is fitted, for a variety of different wind fields. Our emulator for the ocean simulator was fitted to data for only a limited range of input conditions, and so we first screen the data to find only those real storms which lie within the range where we expect reasonable performance from this model; after screening, we find that only 1095 measurements remain.



(a)



(b)

Figure 2.6: Plots of the full emulator fit to the ocean wave simulator (Section 2.6.2): Figure 2.6(a) shows the mean predictions $\mathrm{E}_F[f_i(\theta)]$ and error bars $\mathrm{E}_F[f_i(\theta)] \pm 3\mathrm{Var}_F[f_i(\theta)]^{1/2}$ generated from the fitted emulator (vertical axis) against the true simulator output (horizontal axis) for the validation set of 500 points; Figure 2.6(b) shows the corresponding standardised residuals. Again, the colour of the bars represents the platform to which the prediction corresponds.

For the basis and covariance functions, we choose the same specifications as when fitting the emulator (Section 5.3.2), dropping any dependence on the a. We fix the simulator parameters a to the setting $a^* = \{1.92, 0.391, 2.52, 2.49, 0\}$, since this setting is believed by the model developers to give an acceptable match to the system. To fit the model, we use the same procedure as for the emulator. First, we split the data, and we use a sample of 250 points to carry out an initial regression, to fix the prior moments of the coefficients; the data for this regression are obtained by subtracting the mean of the emulator from the system measurements. Having fixed $\mathrm{E}\left[\beta^{(\delta)}\right]$ and $\mathrm{Var}\left[\beta^{(\delta)}\right]$, we compute the residuals for the regression fit, and fix the marginal covariance of the residual process to the empirical covariance matrix of the residuals. We have $\mathrm{Cov}\left[r_i^{(\delta)}(b), r_j^{(\delta)}(b)\right] = \hat{V}_{ij}$

$$\hat{V}_{ij} = s_i s_j \mathrm{Corr}\left[R_i, R_j\right]$$

where this time, the empirical residual standard deviations are

$$s = (0.6944, 1.2361, 0.7539, 0.4414, 0.1937)^{\mathrm{T}}$$

and the empirical residual correlation matrix is

$$\mathrm{Corr}\left[R_i, R_j\right] = \begin{pmatrix} 1.0000 & 0.9778 & 0.8057 & 0.9231 & -0.0995 \\ 0.9778 & 1.0000 & 0.8108 & 0.9113 & 0.1049 \\ 0.8057 & 0.8108 & 1.0000 & 0.8702 & -0.0032 \\ 0.9231 & 0.9113 & 0.8702 & 1.0000 & -0.0745 \\ -0.0995 & 0.1049 & -0.0032 & -0.0745 & 1.0000 \end{pmatrix}.$$

We see that the residual process for the discrepancy has much the same marginal structure as that of the emulator.

Having fixed the regression prior and the marginal covariance of the residual process, we use the data to estimate the parameters of the correlation function. We employ the same procedure as in the fitting of the emulator. We temporarily fix the emulator and the prior regression surface to their respective mean levels, and for the sample of 795 points which we are using to update the emulator, we subtract these from the measured values; we then carry out the numerical leave-one-out cross validation procedure for the resulting sample of residuals. Treating the emulator and prior

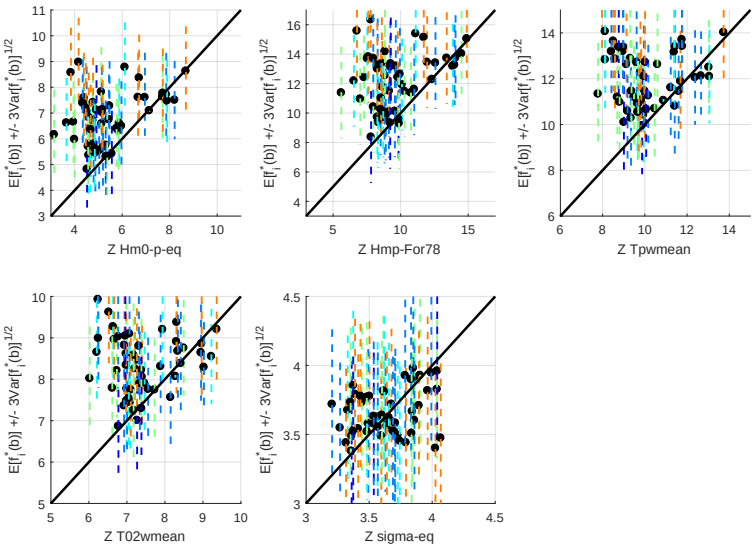
regression surfaces as fixed allows us to temporarily ignore the interactions between these and the residual when approximately determining the correlation parameters, allowing us access to the fast expressions for the predictive mean of each individual point, given the remainder (Rasmussen and Williams [2006], Chapter 5).

We run the cross-validation procedure for a Latin hypercube of 1500 different correlation parameter settings, using the log-Gaussian predictive likelihood as a criterion for comparing different settings; we choose the setting which minimises this criterion, and then jointly update the emulator and the discrepancy model here, as described at 2.4.2. Figure 2.7 demonstrates the update procedure for the discrepancy model. Figure 2.7(a) compares predictions from the emulator for the simulator values (vertical axis) with the true measured value (horizontal axis) for a validation set of 50 points; we see that the predictions for the ocean model tend to be over-estimates of the measured system value, particularly for low values of the first four output components. Figure 2.7(b) then compares the predictions from the full system model (emulator and discrepancy, jointly updated; vertical axis) with the true measured values (horizontal axis) for the same set of 50 points. We see that including the discrepancy in the model and using the system data to learn about it has the effect of improving the quality of our uncertainty specification for the system at points where we have not yet measured the system.

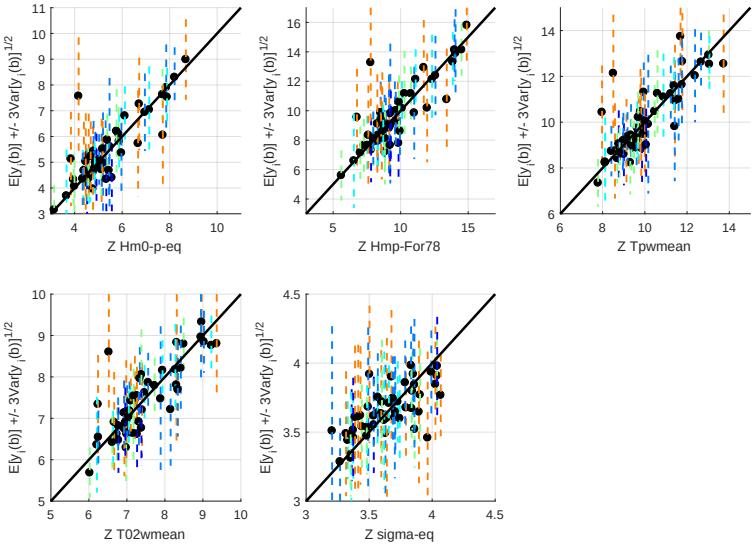
2.6.4 Discussion

In this section, we considered an application of the uncertainty analysis methodology presented in Sections 2.3.1 and 2.4.2 to a particular example, where we must predict the properties of storms from knowledge of certain atmospheric information. An expensive ocean simulator is first emulated as a function of some storm covariates and simulator calibration parameters, and then a model for the discrepancy is posed and updated (jointly with the emulator) using observations of real storms, at a particular, fixed setting of the hindcast calibration parameters.

A few different issues presented themselves during this work, which could be the focus of future work on this problem. First, it was necessary to simplify our treatment of our uncertainty about the hindcast calibration parameters a because, for



(a)



(b)

Figure 2.7: Updating the discrepancy model: Figure 2.7(a) shows the mean predictions $\mathrm{E}\left[\hat{f}_i(b)\right]$ (markers) and error bars $\mathrm{E}\left[\hat{f}_i(b)\right] \pm 3\mathrm{Var}\left[\hat{f}_i(b)\right]^{1/2}$ (dashed lines) under the emulator (vertical axis) against the true measured values z_{ik} (horizontal axis); Figure 2.7(b) shows the mean predictions $\mathrm{E}\left[y_i(b)\right]$ and error bars $\mathrm{E}\left[y_i(b)\right] \pm 3\mathrm{Var}\left[y_i(b)\right]^{1/2}$ for the system (vertical axis) against the measured values (horizontal values) after joint updating of the emulator and the discrepancy as described at 2.4.2. As in Figures 2.5 and 2.6, the colours correspond to the platform.

operational reasons, we were unable to perform future runs on the hindcast having carried out an initial wave of analysis, meaning that a history match (Section 2.4.4) was not possible. If further access to the hindcast was possible in the future, then we could re-focus our emulators on regions of the parameter space which give an acceptable match to the system data, reducing our uncertainty about the simulator in these regions. We could then propagate our uncertainty about these inputs through the emulator by sampling values within our non-implausible space.

Chapter 3

Bayesian optimal design

Models are sometimes developed because they are interesting in their own right, but often, they are developed in order to inform decisions which must be made about the systems that they represent. When faced with a decision about a complex system, Bayesian decision analysis provides us with a general framework in which we can specify all of the possible decisions that we might make, and their consequences under different realisations of the system, before using this information, in conjunction with our inferences about the system, to compare the quality of different decisions against each other.

Once we have specified both a model and a decision problem, there will typically be parameters in our model which we must specify before we can perform an experiment, collect data and perform an inference. This provides us with the opportunity to tune our collection of data on the system in order to increase the likelihood of being able to make a good decision once we have obtained data and computed our posterior beliefs. This is a problem in Bayesian optimal design for which we will develop methods in this chapter.

In Section 3.1, we consider a general Bayesian decision analysis, and discuss some of the computational complexities that the relatively simple written form of the calculations can mask; then in Section 3.2, we consider the design calculations that arise from this decision calculation, and discuss some of the additional computational issues encountered and simple diagnostics that can be used. In Section 3.3, the decision and design frameworks are extended to the situation in which multiple

different experiments are available to us in stages, and at each stage, we must choose whether to stop and make an immediate decision, or to continue sampling in the hope that our knowledge about the system will improve so that we may make better decisions in the future.

In Section 3.4, we develop an approximation procedure for the numerical calculations which draws on some of the analyses discussed in the previous chapter; this framework is designed so as to allow us to track the uncertainty on the individual numerical calculations through the entire problem, to allow us to assess our degree of confidence in the decisions that we make. We illustrate the procedure as it is presented through application to a simple problem, and we go on to consider a more complex application in Section 4.

3.1 Making decisions

Suppose that, after careful consideration, all stakeholders in a given problem have agreed that they wish to make decisions $a \in \mathcal{A}$ about a system which is described by the parameter vector $q \in Q$, and have also specified that the consequences of these decisions under any possible state of the system are described by the loss function $L(a,q) : \mathcal{A} \times Q \to \mathbb{R}$. Consequences of a particular decision/parameter combination can also be expressed in terms of a utility function $U(a,q) = -L(a,q)$, but we choose to work in terms of losses for the remainder of this thesis.

Then, if beliefs about the parameters are described by probability distribution $p(q|\phi)$, given conditioning parameters ϕ which determine the characteristics of the distribution, the optimal decisions are those which minimise the expected loss

$$a^*(\phi) = \arg\min_a \int_Q L(a,q)\, p(q|\phi)\, dq$$

or equivalently, if we were to formulate the problem in terms of utility

$$a^*(\phi) = \arg\max_a \int_Q U(a,q) p(q|\phi)\, dq\ .$$

Assuming that we will always choose to make the optimal decision, we also define the risk from an optimal decision as

$$\rho[\phi] = \min_a \int_Q L(a,q)\, p(q|\phi)\, dq\ . \tag{3.1.1}$$

These calculations are completely general, and extremely simple to express. However, for general posterior distributions and loss functions, it can be very difficult to compute a^*, and by extension, $\rho[\phi]$. The integral in (3.1.1) can only be computed exactly for certain combinations of distribution and loss, and even if this integral can be computed, it is generally not the case that we can perform the optimisation over decisions analytically. Some common, simple cases in which we can compute the optimal decision and the corresponding risk are presented in Section 3.1.1; if we don't wish to operate under these highly restrictive conditions, however, then we must resort to numerical evaluation of the integral and numerical optimisation routines.

3.1.1 Loss functions

We consider some particular loss functions which are widely applied in the literature. These are generally popular for one (or both) of two reasons: either they give rise to a particularly simple form for the risk (under particular/general choices of distribution), or the decisions and losses that they describe are meaningful in a wide variety of problems.

Quadratic loss For a vector of parameters $q = (q_1, \ldots, q_{n_q})^{\mathrm{T}}$, the weighted quadratic loss is defined as [Chaloner and Verdinelli, 1995]

$$L(a, q) = (q - a)^{\mathrm{T}} W(q)(q - a)$$

where $W(q)$ is a symmetric, positive-definite weight matrix. If we differentiate the integral in the risk (3.1.1) with respect to a and set to zero, we find that

$$\bar{W} a^* = \int_Q (W(q)\, q)\, p(q|\phi)\, dq$$

where $\bar{W} = \mathrm{E}[W(q)|\phi]$ is the expectation of the weight matrix over q; the optimal decision is therefore

$$a^* = \bar{W}^{-1} \mathrm{E}[W(q)\, q|\phi] \ .$$

If we substitute this optimal action back into the expression for the risk, we obtain

$$\rho[\phi] = \mathrm{E}\left[q^{\mathrm{T}} W(q)\, q|\phi\right] - \mathrm{E}[W(q)\, q|\phi]^{\mathrm{T}} \bar{W}^{-1} \mathrm{E}[W(q)\, q|\phi] \ .$$

If the elements of the weighting matrix are polynomial functions of the model parameters, then this risk can be evaluated in closed form using the moments of the distribution $p(q|\phi)$.

n-way table For a scalar parameter q, we can also assume a loss function of the form given in table 3.1; we divide the parameter space Q into disjoint segments divided by limits t_j, $j = 1, \ldots, m$, and when the value of q falls within segment $j = 0, \ldots, m$, the consequences of choosing action a_i from a discrete set $\mathcal{A} = \{a_1, \ldots, a_n\}$ are given by the scalar C_{ij}.

Under this loss function, the integral in (3.1.1) can be evaluated easily if the form of the cumulative distribution is known, or if it is simple to evaluate numerically; we have that

$$\rho[\phi] = \min_{a_i} \left[C_{i0} P(q \leq t_1|\phi) + \sum_{j=1}^{m-1} C_{ij} P(t_j \leq q < t_{j+1}|\phi) + C_{im} P(t_m \leq q|\phi) \right] \tag{3.1.2}$$

where

$$P(t_l \leq q < t_r|\phi) = \int_{t_l}^{t_r} p(q|\phi)\, dq \ .$$

There is no general method for optimising the expression (3.1.2) over actions a_i; however, even for a large number of decisions, this discrete optimisation problem is simple in comparison to the continuous optimisation problem which we must solve numerically for general problems with continuous decision spaces. The fact that we must only specify a discrete set of decisions and corresponding consequences may also make the parameters of this problem simple to elicit from an expert.

If we are using a complex loss function or a complex probability distribution and we find that the computational effort required to evaluate the integral or the optimisation in the risk calculation (3.1.1) is too great, then the loss function in table 3.1 can also be used as the basis for an approximation strategy. If we impose a grid in (a, q)-space and approximate the loss function as constant in each cell, then we can compute the expected loss as above; this only requires us to repeatedly evaluate the CDF of the distribution, something which is simple for a fairly wide class of distributions. In doing this, we have replaced the continuous optimisation problem with a much simpler discrete one. A similar approximation can be used

$L(a, q)$	$q \leq t_1$	$t_1 \leq q < t_2$	$\cdots$	$t_m \leq q$
a_1	C_{10}	C_{11}	$\cdots$	C_{1m}
a_2	C_{20}	C_{21}	$\cdots$	C_{2m}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
a_n	C_{n0}	C_{n1}	$\cdots$	C_{nm}

Table 3.1: n-way table loss function (see Section 3.1.1).

for multi-dimensional decision problems, though achieving a good approximation through gridding is harder in higher-dimensional spaces.

3.1.2 Risk under a Bayes linear model

If we do not wish to make a full probabilistic specification for the quantities in a given problem, we may instead choose to make a second-order specification and perform a Bayes linear analysis (as described in chapter 2.1.2); in this instance we do not automatically have access to a probability distribution for q with which to evaluate the integral (3.1.1).

For problems in which the loss function only contains linear and quadratic terms, then we can simply compute the expectation of the loss function directly using our second-order specification; however, in problems where the loss function depends on parameter moments of higher orders or other non-linear terms, then we must specify the higher order moments of the parameters in order to be able to evaluate the expected loss. If information about the form of the higher order moments exists (for example, from Bayes linear variance learning: see Goldstein and Wooff [2007], Chapter 8), then this can be incorporated; a more common strategy is simply to assume a probability distribution with first and second moments which match the Bayes linear specification, and then use this to evaluate (3.1.1).

If we use the weighted quadratic loss (Section 3.1.1) with a constant weight matrix, then we may evaluate the risk exactly using only the first- and second-order moments from the Bayes linear specification; for higher-order polynomials, we must specify at least those moments which are needed to evaluate the expectations of the higher-

order terms, based on the Bayes linear specification. If, however, we use the n-way table, we must specify a probability distribution using our second-order moments in order to be able to evaluate the expression for the risk.

3.2 Design calculations

In many problems, while we may not observe q directly, we may perform an experiment whose outcome depends on its value, and then use Bayes theorem to update beliefs about q before making decisions about its value. We therefore sub-divide the conditioning set ϕ of the general probability distribution $p(q|\phi)$ to reflect the experiment that we will perform in order to learn about the parameters. Following Raiffa and Schlaifer [1961], we subdivide the conditioning set as $\phi = \{z, w, d\}$, where

- $d = \{d_1, \ldots, d_{n_d}\}$ is a set of **design inputs** which must be selected before an experiment can be carried out. For example, in simple cases, d may be an input that specifies which of a discrete number of experiments we will perform, whereas in more complex ones, it may be a set of continuous parameters which govern the location or time at which a measurement is made;

- $w = \{w_1, \ldots, w_{n_w}\}$ is a set of **external** or **environmental inputs** which affect the likely outcomes of the experiment in a way determined by the system model, but which cannot be fully controlled- though the choice of design parameters may alter their distribution $p(w|d)$;

- $z = \{z_1, \ldots, z_{n_z}\}$ are the **observations** that are to be collected on the system, which we will use to learn about the parameters q. In order for the experiment to be a useful one, our model must specify that the distribution of the data is a non-trivial function of q.

within a probabilistic framework, before we carry out an experiment, we specify our prior beliefs $p(q)$ about the parameters (we assume that these are independent of $\{w, d\}$) and we specify a model $p(z|q, w, d)$ which describes the distribution of the data z for given q, w and d. We then carry out the experiment and observe z, at

which point we can compute our updated beliefs about q in the light of this data using Bayes theorem

$$p\left(q|z,w,d\right) = \frac{p\left(z|q,w,d\right)p\left(q\right)}{p\left(z|w,d\right)} \ .$$

We then make decisions about q based on our posterior beliefs about the value of q using equation (3.1.1)

$$\rho\left[z,w,d\right] = \min_{a} \int_{Q} L\left(a,q\right)p\left(q|z,w,d\right)dq \ . \tag{3.2.3}$$

We have control over the value of d at which we carry out the experiment, and through the dependency of $p\left(w|d\right)$ on d, we have some control over the value of w which we are likely to get; it is therefore natural to ask how d might be chosen so as to minimise the risk from experimentation. We must also take into account that different experiments are likely to have different costs $c\left(d\right)$ (also in utility units). This is the design question.

To answer it, we must first take account of the fact that we do not know the value of the data or the environmental parameters before we carry out the experiment; we do this by computing the expectation of the risk over the conditional distributions of both to obtain

$$\bar{\rho}\left[d\right] = \iint \rho\left[z,w,d\right]p\left(z|w,d\right)p\left(w|d\right)dzdw \ . \tag{3.2.4}$$

We now optimise this to find the design with the lowest risk

$$d^{*} = \arg\min_{d}\left[\bar{\rho}\left[d\right] + c\left(d\right)\right]$$

where the corresponding risk is denoted by $\rho^{*} = \bar{\rho}\left[d^{*}\right] + c\left(d^{*}\right)$.

3.2.1 Implementation issues

While it is simple to describe the steps of the optimal design calculation in the general case, when implementing such an analysis in practice, all of the usual computational difficulties associated with a fully Bayesian analysis are magnified.

Computing the risk (3.2.3) from an optimal terminal decision requires us first to perform an integral involving the posterior distribution and loss function; unless we

are willing to assume conjugate forms for the prior distribution and the likelihood for q, leading to a known form for the posterior, and to assume a simple, tractable form for the loss function, such as those discussed in 3.1.1, then we will instantly run into problems. If we are not willing to specify such simple forms, then we must resort to numerical methods (e.g. MCMC) to evaluate the integral, and to numerical optimisation routines to find the optimal decision; this is already a large computational burden, since in general, the integral must be evaluated numerically for each proposed setting of a.

Even making conjugacy assumptions and choosing a simple loss will not generally save us from tractability issues when we come to consider the next stage of the calculation; we must compute the expectation of the risk (which we may only evaluate numerically, possibly at great expense) over the distribution of the data for given design and environmental parameters, and then over the distribution of the environmental parameters given the design. If the posterior $p(q|z, w, d)$ is complex, it is usually as a result of choosing a non-conjugate, potentially non-linear conditional distribution $p(z|q, w, d)$ for the data, and so computing the expectation of the risk over the data is likely to be an even more difficult computational challenge than those encountered so far, and the expectation over the environmental parameters is just the icing on an already very hard-to-swallow cake. It is clear, therefore, that an approximation procedure is going to be necessary for problems of this type.

There is a well-developed literature considering design problems. Raiffa and Schlaifer [1961] is an excellent early piece of work in this area; they present a rich variety of worked examples, restricting themselves to loss functions and distributions which give closed-form risks (in some cases after considerable analytical effort). Chaloner and Verdinelli [1995] also review some linear and simple non-linear design problems, again restricting consideration to problems in which we can derive a closed-form expression for the risk, or for an approximation to the risk. Simulation-based approaches to identifying optimal designs and evaluating the corresponding risk are considered by a number of authors: for example, Clyde et al. [1996] and Muller [1998] adopt an MCMC approach, defining a 'dummy' probability distribution based on the risk and exploring the design space stochastically, and Huan and Marzouk

[2013] develop a framework which uses polynomial chaos expansions as surrogates for highly non-linear models and adopts a stochastic approximation approach to the design calculations themselves.

3.2.2 Value of information

In order for risks such as those in equations (3.2.3) and (3.2.4) to have meaning, they must be placed in the context of other costs associated with the problem at hand: if we perform this calculation and find an optimal design with a risk of 13.2, is this a good or a bad thing? In order to answer this question, there are a number of quantities that we may compute.

Value of sample information We can define a measure of how much we have gained by performing a given experiment by considering the difference between the risk from acting optimally under the prior and the risk from the decision which turns out to be optimal after the experiment. Define

$$a_0 = \arg\min_{a \in \mathcal{A}} \int L(a, q)\, p(q)\, dq$$

as the optimal action under the prior distribution. Then, the *conditional* value of sample information (CVSI; Raiffa and Schlaifer [1961]) for an experiment parametrized by d, which results in environmental parameters w and data z is

$$v(z, w, d) = \min_{a \in \mathcal{A}} \int \left[L(a_0, q) - L(a, q) \right] p(q|z, w, d)\, dq \ .$$

For a given experimental design, we can then compute the *expected* value of sample information

$$v(d) = \iint v(z, w, d)\, p(z|w, d)\, p(w|d)\, dz dw \ .$$

The value of a design d is then $v(d) - c(d)$ - the expected reduction in the risk that is obtained by sampling, minus the cost of performing the experiment at d.

Value of perfect information In a similar way, we can compute an upper bound on the value of any experiment by considering the theoretical experiment which gives

us perfect information about the model parameters q. If q is known exactly, then we can find the optimal decision in any given situation

$$a^*(q) = \arg\min_{a \in \mathcal{A}} L\left(a, q\right) \ .$$

The *conditional* value of perfect information is then the difference between the loss incurred by making the ideal decision and that incurred by making the optimal decision under the prior distribution [Raiffa and Schlaifer, 1961] for a particular value of q

$$w\left(q\right) \ = L\left(a_0, q\right) \ - L\left(a^*(q), q\right) \ .$$

The *expected* value of perfect information is then the expectation of this conditional value over our prior beliefs about q

$$w^* = \min_{a \in \mathcal{A}} \int \left[L\left(a_0, q\right) \ - L\left(a^*(q), q\right) \right] p\left(q\right) \ .$$

The EVPI w^* tells us the value of an experiment in which we would learn the value of q exactly; therefore, there is no real experiment for which we could achieve a lower risk. We can use this value to perform a simple screening of potential experiments. If we find that

$$c\left(d\right) \ \geq w^*$$

for an experiment parametrised by d, then we know without further analysis that the experiment is not worth performing, since the reduction in risk from carrying out the experiment can never be great enough to offset the cost of experimentation.

3.2.3 Simple example

We perform the design calculations for a linear model in order to demonstrate some of the difficulties that we will encounter in even simple examples; each data point is assumed to be the sum of a linear combination of basis functions (with unknown weights) and an error term

$$z_j = g\left(w_j, d_j\right)^{\mathrm{T}} q + \epsilon_j$$

where $g\left(w_j, d_j\right) \ = \left(g_1\left(w_j, d_j\right), \ldots, g_{n_q}\left(w_j, d_j\right)\right)^{\mathrm{T}}$ is a vector of known functions, and $q = \left(q_1, \ldots, q_{n_q}\right)^{\mathrm{T}}$ is the vector of parameters about which we wish to learn. For the

full data vector, our model is

$$z = G(w,d)\, q + \epsilon$$

where $G(w,d) = \left(g(w_1,d_1), \ldots, g\left(w_{n_z},d_{n_z}\right)\right)^{\mathrm{T}}$ is the usual regression design matrix and $\epsilon = (\epsilon_1, \ldots, \epsilon_{n_z})^{\mathrm{T}}$ is the corresponding vector of errors. We make a Gaussian specification for both the prior $p(q)$ and the error distribution

$$q \sim \mathcal{N}(\mu_q, V_q) \qquad\qquad \epsilon \sim \mathcal{N}\left(0, \frac{1}{\lambda_\epsilon} I\right) .$$

Making a Gaussian specification for the prior and the error structure ensures that all of relevant predictive, marginal and posterior distributions are also Gaussian; for our design calculations, we require the marginal distribution of the data $p(z|w,d)$, which is simply

$$z|w,d \sim \mathcal{N}(\mu_z(w,d), V_z(w,d))$$

where

$$\mu_z(w,d) = G(w,d)\,\mu_q \qquad V_z(w,d) = G(w,d)\, V_q G(w,d)^{\mathrm{T}} + \frac{1}{\lambda_\epsilon} I .$$

After observation of the data, our posterior beliefs about the parameters are summarised by another Gaussian distribution

$$q|z,w,d \sim \mathcal{N}\left(\hat{\mu}_q(z,w,d), \hat{V}_q(w,d)\right)$$

where the parameters are

$$\begin{aligned} \hat{V}_q(w,d) &= \left[V_q^{-1} + \lambda_\epsilon G(w,d)^{\mathrm{T}} G(w,d)\right]^{-1} \\ \hat{\mu}_q(z,w,d) &= \hat{V}_q(w,d)\left[V_q^{-1}\mu_q + \lambda_\epsilon G(w,d)^{\mathrm{T}} z\right] . \end{aligned} \tag{3.2.5}$$

Let us now assume that our decision problem is specified through a weighted quadratic loss function (see Section 3.1.1), with constant, symmetric, positive-definite weight matrix W; the resulting risk under the posterior is then [Chaloner and Verdinelli, 1995]

$$\rho[z,w,d] = \rho[w,d] = \mathrm{tr}\left(W\hat{V}_q(w,d)\right) .$$

The risk does not depend on the observed values of the data in this case, owing to the fact that the posterior variance also does not depend on z. Even with this property,

deriving a closed-form expression for this risk will be difficult for high-dimensional parameter vectors, since we must invert a dense matrix ((3.2.5)) which is a function of both the design and environmental inputs in order to do so.

Specific example To bypass some of the numerical difficulties presented by even this problem, we consider the problem in which the data does not depend on any environmental parameters w, in which our basis vector is

$$g\left(d_j\right) = \begin{pmatrix} 1 \\ (1+d_j)(1-d_j) \end{pmatrix}$$

and our prior covariance matrix for the parameters q is $V_q = \mathrm{Diag}\left(1/\lambda_{q_1}, 1/\lambda_{q_2}\right)$. Using this vector of regressors, the resulting posterior variance is

$$\hat{V}_q\left(d\right) = \frac{1}{D_q\left(d\right)} \begin{pmatrix} \lambda_{q_2} + \lambda_\epsilon \sum_j (1+d_j)^2(1-d_j)^2 & -\lambda_\epsilon \sum_j (1+d_j)(1-d_j) \\ -\lambda_\epsilon \sum_j (1+d_j)(1-d_j) & \lambda_{q_1} + \lambda_\epsilon n_z \end{pmatrix}$$

where $D_q\left(d\right)$ is the determinant of the inverse matrix

$$D_q\left(d\right) = (\lambda_{q_1} + \lambda_\epsilon n_z)\left(\lambda_{q_2} + \lambda_\epsilon \sum_j (1+d_j)^2(1-d_j)^2\right) - \left(\lambda_\epsilon \sum_j (1+d_j)(1-d_j)\right)^2 .$$

From this, we can obtain an expression for the risk by multiplying by the weight matrix W and taking the trace.

Removing any dependence on the environmental parameters and assuming a form for the loss which results in a data-independent risk turns computing the expectation of the risk over these two components into a triviality; we can see through the resulting non-linear dependence of the risk on d though that such expectations could not have been computed analytically anyway. Even after simplifying our problem to allow us to progress this far, we arrive in a situation where we must resort to numerical optimisation techniques in order to find the optimal design and the corresponding risk.

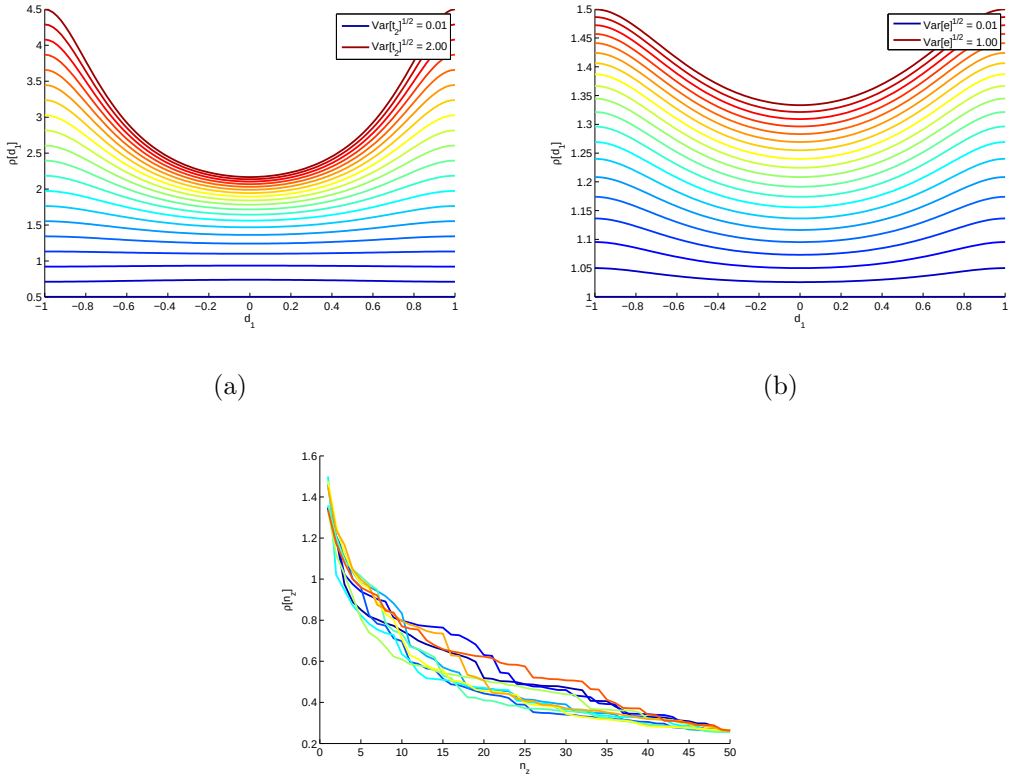
Figure 3.1 shows the risk function plotted at different values of various parameters; all plots are generated for $W = I$. Figure 3.1(a) shows the risk $\rho\left[d_1\right]$ for an experiment where we make only a single observation z_1; the different coloured lines correspond to different settings of the prior variance $\mathrm{Var}\left[q_2\right]$ of the second model

parameter. Because of the symmetry of both $g(.)$ in their input arguments, our minimum risk either lies in the centre of the input domain ($d_1 = 0$) or at its edges ($d_1 = 1$ or $d_1 = -1$). Figure 3.1(b) shows the same risk function, but with the different colours representing different values of $\mathrm{Var}[\epsilon]$. Figure 3.1(c) shows the behaviour of the risk as we acquire observations $z_1, \ldots, z_{50}$ incrementally, calculating the risk after each data point is acquired; the different colours correspond to different space-filling samples of $d = \{d_1, \ldots, d_{50}\}$. We see that the decrease in the risk as we acquire the first few observations is relatively sharp, with the rate of decrease slowing as each data point makes less of a difference to our state of knowledge about q.

3.3 Sequential design calculations

In many real-world data collection problems, we have the option to perform multiple experiments to learn about the same set of parameters. An obvious benefit of performing multiple experiments in any case is that each will decrease our uncertainty about the parameter values, though this benefit must be offset against the costs of performing the experiments. In the case where the experiments are to be performed in sequence, the results of the experiments which we have performed so far will be available to us at the point where we must decide whether to commission further sampling; in this situation, we can design the next experiment against our posterior beliefs given the data obtained so far, enabling us to maximise the value of the information that we will collect in the future.

Consider the following example: we are collecting observations of the atmospheric concentration of a particular gas species in order to infer the locations and emission rates of gas sources in a given region. Before any data has been observed, we believe that sources are equally likely to be anywhere in the region. Suppose that we make a first set of concentration observations, and we consider the effect of these observations on our beliefs; we find that there is an elevated concentration in measurements made when the wind is blowing from a particular arc of directions. If we now have the opportunity to re-locate our measuring equipment, then we may



(a) (b)

(c)

Figure 3.1: Plots of the risk function from Section 3.2.3 for different parameter settings: Figure 3.1(a) shows $\rho\left[d_1\right]$ for a single observation, with the colour scale varying over $\mathrm{Var}\left[q_2\right]$; Figure 3.1(b) shows the same risk, with the colours representing different settings of $\mathrm{Var}\left[\epsilon\right]$. Figure 3.1(c) shows the change in the risk as we incrementally acquire data $z_1, \ldots, z_{50}$; the different coloured lines represent different space-filling choices of d.

use the information that we have already obtained to help select a location; if the signal in the data seems to be coming from a particular part of the domain, then moving the sensors downwind of this region may improve the information content of the data that we collect in the future, and potentially allow us to more accurately determine the locations and emission rates of sources.

While re-designing after data has been observed in this way has potential benefits, it also presents a large computational challenge when applied to a general problem. When selecting design parameters at early stages of the problem, we need to consider the effects of our design choices now on the value of the information that we might collect in the future, making this problem exponentially more complex than the single-experiment design problem outlined in Section 3.2. In the remainder of this section, we outline the calculations that we need to perform in order to find the optimal sequential design; Section 3.3.1 introduces the problem and notation, and Section 3.3.2 outlines the calculations that we need to perform in order to select a design. In Section 3.3.3, we relate the computational burden of this calculation to that of those that we have already encountered, and motivate the need for an approximating framework.

3.3.1 Problem specification

We begin by introducing a notation and formally specifying the problem to be solved: our treatment of the sequential problem follows that of DeGroot [1970]. We assume that there is a sequence of n possible experiments that we can perform, which we index by j, each of which allows us to collect data which is informative for q. We specify our prior beliefs about q, and we assume that the forward model for each of the experiments has been specified.

Before the observations from the j^{th} experiment can be collected, a corresponding set of design parameters must be specified; each set of observations has an associated cost (possibly depending on the setting of the design parameters). After the observations have been collected, we have a choice: we may make an immediate decision, based on our current belief state, which minimises our expected loss; alternatively, we may choose to perform the $(j+1)^{\text{th}}$ experiment (for which we must select a

design), after which we will be faced with the same choice between an immediate decision and further sampling. As in the non-sequential problem, we assume that our beliefs about the data will be affected by a set of external parameters, which will be unknown to us at the point at which we must select our design.
Our goal is to be able to determine, for any state that we find ourselves in, whether it is optimal to take the next set of observations, or to make an immediate decision based on our current beliefs. We introduce the following notation for the sequential problem:

- We denote the data available from experiment j by $z_j = \{z_{j1}, \ldots, z_{jn_{z_j}}\}$, and we denote the set of all data available up to and including experiment j by $z_{[j]} = \{z_1, \ldots, z_j\}$;
- We denote the external inputs which affect the j^{th} experiment by $w_j = \{w_{j1}, \ldots, w_{jn_{w_j}}\}$, and the collection of all such parameters up to and including experiment j by $w_{[j]} = \{w_1, \ldots, w_j\}$;
- We denote the design inputs which control the j^{th} experiment by $d_j = \{d_{j1}, \ldots, d_{jn_{d_j}}\}$, and the collection of all of these up to and including experiment j by $d_{[j]} = \{d_1, \ldots, d_j\}$;
- If making a decision immediately after experiment j, we must select actions $a_j \in \mathcal{A}_j$, and the losses that we incur are described by the loss function $L_j(a_j, q)$. The optimal decision that we make immediately after experiment j is denoted by a_j^*;
- The cost of choosing design setting d_j for experiment j is given by the cost function $c_j(d_j)$.

After having carried out the experiments $1, \ldots, j$, at design parameter settings $d_{[j]}$ and under external inputs $w_{[j]}$, our posterior beliefs about the parameters q are

$$\begin{aligned} p\left(q|z_{[j]}, w_{[j]}, d_{[j]}\right) &= \frac{p\left(z_{[j]}|q, w_{[j]}, d_{[j]}\right) p\left(q|w_{[j]}, d_{[j]}\right)}{p\left(z_{[j]}|w_{[j]}, d_{[j]}\right)} \\ &= \frac{p\left(z_{[j]}|q, w_{[j]}, d_{[j]}\right) p\left(q\right)}{p\left(z_{[j]}|w_{[j]}, d_{[j]}\right)} \end{aligned}$$

where, as in the single-experiment design problem, we assume that our prior beliefs about q do not depend on $\{w_{[j]}, d_{[j]}\}$. In general, we assume that our beliefs $p\left(w_{[j]}|d_{[j]}\right)$ about the external parameters depend on the setting of the design inputs; we also assume that we do not use the data $z_{[j]}$ to learn about the external parameters.

In this thesis, we consider only bounded sequential decision problems; that is, problems in which there is a maximum number n of experiments which can be performed. We will see in Section 3.3.2 that we can develop an algorithm which computes the risk from an optimal course of action for general bounded problems; due to the lack of a final experiment at which to begin, we cannot develop such an algorithm for general unbounded problems. It is, however, possible to find solutions for some particular, simple, unbounded problems; see, for example, Berger [1985]. If an unbounded problem cannot be solved exactly, then it is usually approximated by a bounded problem with a large number of stages, and the sensitivity to the number of stages used in the approximation is assessed. For further discussion of unbounded problems, see, for example, the books by Berger or DeGroot [1970].

3.3.2 Backward induction

We begin by considering whether the first set of observations should be taken (at some design setting), or whether a decision should be taken immediately, based on our prior beliefs. If these were the only observations that we could make, then we would proceed as in Section 3.2 and evaluate the relative merits of obtaining the data against taking an immediate decision. However, in the sequential problem, after this initial experiment, we must make another choice: we must decide whether to take a second set of observations, or whether to take a decision based on our posterior beliefs given the first data-set. In order to solve the design problem for the first experiment, then, we must know the risks from deciding to carry out a second experiment for all possible outcomes of the first. By the same argument, we must also know the risks from a third experiment for all possible outcomes of the second in order to solve the design problem for the second, and so on.

This argument is used by DeGroot [1970] to motivate the development of an algo-

rithm which starts from the last possible set of observations which we might make; at this point, further sampling is not possible, and so we must make an immediate decision based on the data that we have obtained so far. From this point, we sequentially perform single-experiment design calculations, as in Section 3.2, comparing the risk from an optimally designed future experimentation procedure with the risk from an immediate decision based on the information available up to the present time. If we choose to stop after experiment j and make an immediate decision, then the resulting risk from an optimal decision (known as the risk from an optimal terminal decision, or the terminal risk) is defined in the same way as in equation (3.2.3)

$$\rho_j^t \left[z_{[j]}, w_{[j]}, d_{[j]} \right] = \min_{a_j \in \mathcal{A}_j} \int L\left(a_j, q\right) p\left(q | z_{[j]}, w_{[j]}, d_{[j]}\right) dq \ . \tag{3.3.6}$$

The backward induction algorithm uses these terminal risks to build a sequence of risk functions $\rho_j\left[.\right]$ which describe the risk as a function of all parameters $\{z_{[j]}, w_{[j]}, d_{[j]}\}$ at the current stage under the assumption that we will act optimally at all points in the future.

In what follows, 'Stage j' of the calculation refers to the point at which $(j-1)$ experiments have been performed, and we must assess the value of the j^{th}. At stage n, the final stage, we perform the usual design calculations as a function of $\{z_{[n-1]}, w_{[n-1]}, d_{[n-1]}\}$:

- After the n^{th} experiment, further samples are not possible, and so our overall risk is the same at the terminal risk

$$\rho_n \left[z_{[n]}, w_{[n]}, d_{[n]} \right] = \rho_n^t \left[z_{[n]}, w_{[n]}, d_{[n]} \right] \ .$$

- The data z_n and the environmental parameter w_n will be unknown at the point where we must select d_n, so we compute the expectation of the risk over these parameters

$$\bar{\rho}_n \left[z_{[n-1]}, w_{[n-1]}, d_{[n]} \right] = \iint \rho_n \left[z_{[n]}, w_{[n]}, d_{[n]} \right] p\left(z_n | z_{[n-1]}, w_{[n]}, d_{[n]}\right) \times p\left(w_n | w_{[n-1]}, d_{[n]}\right) dz_n dw_n \ . \tag{3.3.7}$$

- For given $\{z_{[n-1]}, w_{[n-1]}, d_{[n-1]}\}$, we optimise this risk over d_n in order to find the risk from an optimally-designed experiment at the final stage, taking into account the cost of the experiment

$$\rho_n^* \left[z_{[n-1]}, w_{[n-1]}, d_{[n-1]}\right] = \min_{d_n \in \mathcal{D}_n} \left[\bar{\rho}_n \left[z_{[n-1]}, w_{[n-1]}, d_{[n]}\right] + c_n\left(d_n\right)\right] . \quad (3.3.8)$$

Having identified the risk from an optimally-designed experiment at the final stage as a function of the designs and observed quantities from the previous experiments, we use the risk $\rho_n^*[.]$ to initialise the remainder of the algorithm, carrying out the following steps for stages $j = (n-1), \ldots, 1$:

- After the j^{th} experiment, we find the risk ρ_j from an optimal course of action for any setting of $\{z_{[j]}, w_{[j]}, d_{[j]}\}$ by comparing the risk ρ_j^t from an immediate decision with the risk ρ_{j+1}^* from optimally-designed future sampling

$$\rho_j \left[z_{[j]}, w_{[j]}, d_{[j]}\right] = \min \left[\rho_j^t \left[z_{[j]}, w_{[j]}, d_{[j]}\right], \rho_{j+1}^* \left[z_{[j]}, w_{[j]}, d_{[j]}\right]\right] . \quad (3.3.9)$$

  The risk ρ_n^* from an optimally-designed experiment at the final stage is defined in (3.3.8); for any other stage, the risk ρ_{j+1}^* is defined in (3.3.11).

- We compute the expectation $\bar{\rho}_j$ of this risk over our beliefs about the observed data z_j and environmental parameters w_j at this stage

$$\bar{\rho}_j \left[z_{[j-1]}, w_{[j-1]}, d_{[j]}\right] = \iint \rho_j \left[z_{[j]}, w_{[j]}, d_{[j]}\right] p\left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]}\right) \times p\left(w_j | w_{[j-1]}, d_{[j]}\right) dz_j dw_j . \quad (3.3.10)$$

- We find the optimal design d_j^* for the j^{th} experiment and the corresponding risk ρ_j^* as a function of the risk inputs $\{z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\}$ from the previous stages, taking into account the cost of running the experiment at a particular d_j

$$\rho_j^* \left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\right] = \min_{d_j \in \mathcal{D}_j} \left[\bar{\rho}_j \left[z_{[j-1]}, w_{[j-1]}, d_{[j]}\right] + c_j\left(d_j\right)\right] . \quad (3.3.11)$$

Working backward through these steps, we eventually compute the risk ρ_1^* from an optimally designed experiment at the first stage; this risk takes account of all possible

future outcomes, assuming that we will always design and act optimally based on the information currently available to us. At this point, we face our first real choice, between performing the first experiment and making an immediate decision based on our prior beliefs. Of course, we make this choice by comparing ρ_1^* with ρ_0^t:

- if $\rho_0^t \leq \rho_1^*$, then the costs of collecting the data offset any benefit that we might expect to gain from observing them; in this instance, it is optimal just to make a decision a_0^* now based on our prior beliefs;
- if $\rho_0^t > \rho_1^*$, then our analysis has concluded that it is worth performing the first experiment at d_1^*.

If we conclude that it is optimal to carry out at least the first experiment, then we do so (at d_1^*), collecting $\{z_1, w_1\}$. In the light of this new information, we now assess whether it is optimal to make an immediate decision at this point, or whether we should perform the second experiment; we do this by comparing $\rho_1^t[z_1, w_1, d_1^*]$ with $\rho_2^*[z_1, w_1, d_1^*]$ in the same way. We work forward through the experiments in this way, deciding on the optimal course of action after experiment k by comparing $\rho_k^t\left[z_{[k]}, w_{[k]}, d_{[k]}^*\right]$ with $\rho_{k+1}^*\left[z_{[k]}, w_{[k]}, d_{[k]}^*\right]$, where $d_{[k]}^* = \{d_1^*, \ldots, d_k^*\}$.
The backward induction calculation is also presented as pseudo-code in algorithm 1. Note that if it is possible to compute all of the functions (3.3.6), (3.3.9), (3.3.10) and (3.3.11) without resorting to numerical methods, then we only need to run the algorithm once, as then after each experiment, we can simply substitute the observed values $\{z_k, w_k\}$ and the chosen design d_k^* into the appropriate risk functions, and read off the resulting d_{k+1}^*, ρ_{k+1}^* and ρ_k^t. However, as will be discussed in the following section, this simple situation is extremely unlikely to occur.

3.3.3 The computational burden

It is clear from the form of the backward induction calculation that all of the tractability issues encountered in the non-sequential problem will also be encountered in the sequential problem, but compounded by the recursive nature of the calculation. As was the case for non-sequential problems, it is only for very carefully-chosen priors, likelihoods and loss functions that we can even compute $\rho_n^t[.]$ in

closed-form; even in this situation, we are almost guaranteed to lose tractability at some point during steps (3.3.7) or (3.3.8) at stage n, so comparing risk functions $\rho_{n-1}^{t}[.]$ and $\rho_{n}^{*}[.]$ point-wise as in (3.3.9) will not result in a closed-form expression. If we want to perform this calculation, then, we are going to have to do it numerically; however, it is easy to see how we might run into problems here too. Traditional numerical integration or optimisation techniques (for example, Gauss-Hermite quadrature or the Nelder-Mead simplex algorithm) require us to evaluate the target function at a specially-chosen sequence of points. Applying these methods naively to this problem would result in the number of risk function evaluations required increasing exponentially in the number of stages of the problem: the numerical optimisation over d_j that would be required to evaluate $\rho_{j}^{*}[.]$ would require a separate numerical integral to be performed to evaluate $\bar{\rho}_j$ for each candidate setting of d_j; numerically evaluating $\bar{\rho}_j$ in turn requires ρ_{j+1}^{*} to be evaluated for many different values of $\{z_j, w_j\}$, which calls for for a numerical optimisation to be performed for each such input setting, and so on. This approach would clearly be infeasible in all but the simplest of problems. In Section 3.4, we review some of the ways in which these calculations have been approximated in the past, before going on to develop an approximation which uses Bayes linear emulators to track uncertainty in the numerical calculations that we perform.

3.4 Approximation of the backward induction calculation

Since it looks to be computationally infeasible to perform the calculations in algorithm 1 using traditional numerical techniques, we must develop an alternative means of approximation. An approach which uses second-order emulators would seem to confer a number of advantages in this context:

- The risk functions $\rho_j[.]$ are systematic functions of their inputs, and fitting emulators allows us to exploit this structure in our numerical calculations; if we were instead to use a traditional Monte-Carlo scheme for numerical evaluation,

Algorithm 1 Backward induction algorithm for sequential design problems

1: **for** $k = 0, 1, \ldots, (n-1)$ **do**

2: **for** $j = n, (n-1), \ldots, (k+1)$ **do**

3: Compute the risk

4: **if** $j = n$ **then**

$$\rho_n \left[z_{[n]}, w_{[n]}, d_{[n]} \right] = \rho_n^t \left[z_{[n]}, w_{[n]}, d_{[n]} \right]$$

5: **else**

$$\rho_j \left[z_{[j]}, w_{[j]}, d_{[j]} \right] = \min \left[\rho_j^t \left[z_{[j]}, w_{[j]}, d_{[j]} \right], \rho_{j+1}^* \left[z_{[j]}, w_{[j]}, d_{[j]} \right] \right]$$

6: **end if**

7: Compute the expected risk

$$\bar{\rho}_j \left[z_{[j-1]}, w_{[j-1]}, d_{[j]} \right] = \iint \rho_j \left[z_{[j]}, w_{[j]}, d_{[j]} \right] p \left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]} \right) \times p \left(w_j | w_{[j-1]}, d_{[j]} \right) dz_j dw_j$$

8: Compute the optimal design, and corresponding risk:

$$\rho_j^* \left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]} \right] = \min_{d_j \in \mathcal{D}} \left[\bar{\rho}_j \left[z_{[j-1]}, w_{[j-1]}, d_{[j]} \right] + c_j \left(d_j \right) \right]$$

9: **end for**

10: **if** $\rho_k^t \leq \rho_{k+1}^*$ **then**

11: Cease sampling, and take immediate decision a_k^*

12: **else**

13: Pay observation cost $c_{k+1} \left(d_{k+1}^* \right)$, and observe data $\{ z_{k+1}, w_{k+1} \}$ at d_{k+1}^*.

14: **end if**

15: **end for**

we would throw away some of this structural information, potentially increasing the computer time needed for the calculation (see the second objection in O'Hagan [1987]);

- If we can emulate the risk as a function of its inputs, then we immediately gain access to the tools developed for emulators in Section 2.3; in particular, we can approximate the expectations that we are required to compute in line 7 of the algorithm by integrating the emulator directly, thereby translating our inferences about the function itself directly into inferences about its expected value;

- The global optimisation that we are required to perform in line 8 presents a particular challenge, since standard numerical optimisation techniques are not guaranteed to find a global minimum. We would also run into difficulty if trying to use a Gaussian process to find the probability that any given input setting is the minimum, since the resulting distribution is difficult to write down, and even more difficult to work with (see Henning and Schuler [2012]). Using a second-order emulator has the potential to make things easier; we can develop a procedure similar to history matching which allows us to rule out parts of the input space that are unlikely to be minima, and then try to sequentially reduce the size of this space through further analysis.

The procedure that we propose in the coming sections approximates the risk at each stage of the backward induction calculation using second-order emulators; we then approximate each expected risk by integrating our emulator directly, and use a simple sampling procedure to characterise our uncertainty about each risk at its minimum. We run this procedure in waves, in a similar way to the history matching procedure of Vernon et al. [2010]: at the first wave, we fit emulators that explore the structure of the risk over the whole of the design space, and use these to rule out designs which are unlikely to minimise the risk; then, at later waves, we re-fit our emulators in those parts of the space which have not been ruled out, allowing us to focus our efforts on modelling the risk in those regions which are most interesting. As the volume of the design input domain for our emulator decreases, we can fit a

more accurate emulator within this region, giving us greater power to discriminate between designs on the basis of their risks.

Previous work in this area includes that of Müller et al. [2007]; in their study, they approximate the sequential design problem in relatively low-dimensional cases by gridding the design space at each stage and explore this gridded space by using an MCMC scheme which simulates from the priors. This approach works well in problems with a small number of design parameters which must be found, but quickly becomes difficult when experiments depend on a large number of design inputs, or where a large number of experimental stages are possible. Williamson and Goldstein [2012] provide decision support for policy makers in a climate modelling problem by using emulators to approximate the backward induction calculation which arises.

More recently, Huan and Marzouk [2016] cast the sequential optimal design problem in a dynamic programming framework, and consider the relationship between the full sequential design and simple approximations in which the experiments are simply designed separately ('batch design') or designed in sequence without considering the possibility of future experiments ('greedy design'). They then propose an approximation to the general sequential design problem which fits a regression model to each risk, and uses a 'one step look-ahead' approximation to the full backward induction calculation which only considers the next available experiment.

In Section 3.4.1, we outline the steps of the approximating procedure, linking them to the steps of the original backward induction algorithm, and in Section 3.4.2, we set up a simple problem, which will be used as a running example to illustrate the steps taken throughout the chapter; then, in Sections 3.4.3 to 3.4.10, we consider implementation details for each of the steps of the approximation, and, where appropriate, illustrate them through application to our simple problem.

3.4.1 Approximating procedure

We perform the approximating procedure in waves, indexed by $i = 1, 2, \ldots$; the backward induction algorithm is approximated by iterating the following steps for stages $j = n, (n-1), \ldots, 1$. The approximate backward induction procedure is also presented as pseudo-code in algorithm 2.

Emulate the risk We begin by fitting an emulator $r_j^{(i)}$ to the risk at stage j: this emulator has the common regression plus residual form, with an additional iid 'nugget' term

$$r_j^{(i)}\left[z_{[j]}, w_{[j]}, d_{[j]}\right] = \sum_p \alpha_{jp}^{(i)} h_{jp}^{(i)}\left(z_{[j]}, w_{[j]}, d_{[j]}\right) + u_j^{(i)}\left(z_{[j]}, w_{[j]}, d_{[j]}\right) + \xi_j^{(i)} \quad (3.4.12)$$

where the $h_{jp}^{(i)}(.)$ are known basis functions, and the uncertain components (coefficients $\{\alpha_{jp}^{(i)}\}$, residual process $u_j^{(i)}(.)$ and nugget $\xi_j^{(i)}$) are assumed to be a priori independent. We specify prior moments for the components of the model, and then use a set of evaluations of the risk to adjust these moments.

The generation of the risk evaluations that we use to adjust our prior beliefs is discussed in detail in Section 3.4.4, and such an adjustment is carried out for our simple example in Section 3.4.5. In Section 3.4.11, we discuss the selection of the input settings at which we run the risk to generate the data for the update.

Approximate the expected risk Our approximation to the expected risk $\bar{\rho}_j$ at wave i is simply the integral of the emulator $r_j^{(i)}$

$$\bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}\right] = \iint r_j^{(i)}\left[z_{[j]}, w_{[j]}, d_{[j]}\right] \times p\left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]}\right) p\left(w_j | w_{[j-1]}, d_{[j]}\right) dz_j dw_j \ . \quad (3.4.13)$$

Our beliefs about $\bar{r}_j^{(i)}$ are computed by directly integrating our beliefs about $r_j^{(i)}$ as described in Section 2.4.1; implementation in this context is discussed further in Section 3.4.6. The characterisation of the distributions $p\left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]}\right)$ and $p\left(w_j | w_{[j-1]}, d_{[j]}\right)$ for complex models is discussed in Section 3.4.3.

Characterise the risk from an optimal design Our approximation to the risk ρ_j^* from an optimal design at wave i is denoted by $s_j^{(i)}$, with

$$s_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\right] = \bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, \{d_{[j-1]}, d_j^*\}\right] + c_j\left(d_j^*\right) \ . \quad (3.4.14)$$

The value of d_j^* is unknown; we represent our uncertainty about the optimal design by sampling candidate designs $\tilde{d}_j$ from within a candidate design space $\mathcal{D}_j^{(i)}$ which could plausibly contain the optimal design. Our strategy for sampling the candidate designs and for characterising our beliefs about $s_j^{(i)}$ is discussed in Section 3.4.8.

Algorithm 2 Backward induction approximation algorithm

1: **for** $i = 1, 2, \ldots$ **do**

2: **for** $j = n, (n-1), \ldots, 1$ **do**

3: Specify risk model (Section 3.4.4)

$$r_j^{(i)}\left[z_{[j]}, w_{[j]}, d_{[j]}\right] = \sum_p \alpha_{jp}^{(i)} h_{jp}^{(i)}\left(z_{[j]}, w_{[j]}, d_{[j]}\right) + u_j^{(i)}\left(z_{[j]}, w_{[j]}, d_{[j]}\right) + \xi_j^{(i)}$$

4: Approximate the expected risk (Section 3.4.6)

$$\bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}\right] = \iint r_j^{(i)} p\left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]}\right) p\left(w_j | w_{[j-1]}, d_{[j]}\right) dz_j dw_j$$

5: Characterise the minimum risk (Section 3.4.8)

$$s_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\right] = \bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, \{d_{[j-1]}, d_j^*\}\right] + c_j\left(d_j^*\right)$$

6: **end for**

7: **end for**

3.4.2 Running example

Throughout this chapter, we illustrate some of the approximation steps taken through application to a simple problem; in this problem, the forward model is simply a Gaussian linear model, and we will make only one observation at a time. In this section, we set up the model and derive all of the necessary posterior and marginal distributions, then we set up the decision problem that we will solve for this model, and compute the terminal risk. We will begin the approximate design calculations for a two stage problem in Section 3.4.5.

System model As with the model in Section 3.2.3, the data is represented using a linear combination of basis functions $g(d) = (g_1(d), \ldots, g_{n_q}(d))^{\mathrm{T}}$ with weights $q = (q_1, \ldots, q_{n_q})^{\mathrm{T}}$; we assume that we make a single observation at each stage, and so we have

$$z_j = g(d_j)^{\mathrm{T}} q + \epsilon_j$$

where ϵ_j is a measurement error term. The data are assumed not to depend on any external parameters w_j in this example. Assuming that in this case, $z_{[j]} =$

$(z_1, \ldots, z_j)^{\mathrm{T}}$ is a vector, we have the following for multiple data points

$$z_{[j]} = G\left(d_{[j]}\right) q + \epsilon$$

where

$$G\left(d_{[j]}\right) = \begin{pmatrix} g\left(d_1\right)^{\mathrm{T}} \\ \vdots \\ g\left(d_j\right)^{\mathrm{T}} \end{pmatrix} \qquad \epsilon = \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_j \end{pmatrix} .$$

Assuming as before that our prior beliefs $q \sim \mathcal{N}\left(\mu_q, V_q\right)$ and error distributions $\epsilon_j \sim \mathcal{N}\left(0, \frac{1}{\lambda_\epsilon}\right)$ are Gaussian, our posterior distribution at stage j is

$$q | z_{[j]}, d_{[j]} \sim \mathcal{N}\left(\hat{\mu}_q\left(z_{[j]}, d_{[j]}\right), \hat{V}_q\left(d_{[j]}\right)\right)$$

where

$$\begin{aligned} \hat{V}_q\left(d_{[j]}\right) &= \left[V_q^{-1} + \lambda_\epsilon G\left(d_{[j]}\right)^{\mathrm{T}} G\left(d_{[j]}\right)\right]^{-1} \\ \hat{\mu}_q\left(z_{[j]}, d_{[j]}\right) &= \hat{V}_q\left(d_{[j]}\right)\left[V_q^{-1}\mu_q + \lambda_\epsilon G\left(d_{[j]}\right)^{\mathrm{T}} z_{[j]}\right] . \end{aligned}$$

It is also simple to compute the predictive distribution of a new data point $\hat{z}$ at a new design setting $\hat{d}$ based on these posterior beliefs

$$\hat{z} | \hat{d}, z_{[j]}, d_{[j]} \sim \mathcal{N}\left(\mu_{\hat{z}}\left(z_{[j]}, d_{[j]}\right), V_{\hat{z}}\left(d_{[j]}\right)\right) \tag{3.4.15}$$

where the parameters of the distribution are

$$\mu_{\hat{z}}\left(z_{[j]}, d_{[j]}\right) = g\left(\hat{d}\right)\hat{\mu}_q\left(z_{[j]}, d_{[j]}\right) \qquad V_{\hat{z}}\left(d_{[j]}\right) = g\left(\hat{d}\right)\hat{V}_q\left(d_{[j]}\right) g\left(\hat{d}\right)^{\mathrm{T}} + \frac{1}{\lambda_\epsilon} .$$

Decision problem To maintain the simplicity of the problem, we use the weighted quadratic loss function outlined in Section 3.1.1, and we specify that our loss is based on the prediction $\hat{z}$ for the system data at a new, known design input $\hat{d}$; the weighting function that we use at stage j is chosen to be a second-order polynomial in $\hat{z}$

$$w_j\left(\hat{z}\right) = a_j \hat{z}^2 + b_j \hat{z} + c_j .$$

Using this weighting in the quadratic loss, the resulting closed-form risk at stage j is

$$\rho_j^t\left[z_{[j]}, d_{[j]}\right] = \mathrm{E}\left[w_j\left(\hat{z}\right)\hat{z}^2\right] - \frac{1}{\mathrm{E}\left[w_j\left(\hat{z}\right)\right]}\mathrm{E}\left[w_j\left(\hat{z}\right)\hat{z}\right]^2$$

where

$$\begin{aligned}
\mathrm{E}\left[w_j\left(\hat{z}\right)\hat{z}^2\right] &= a_j m_4\left(\mu_{\hat{z}}, V_{\hat{z}}\right) + b_j m_3\left(\mu_{\hat{z}}, V_{\hat{z}}\right) + c_j m_2\left(\mu_{\hat{z}}, V_{\hat{z}}\right) \\
\mathrm{E}\left[w_j\left(\hat{z}\right)\hat{z}\right] &= a_j m_3\left(\mu_{\hat{z}}, V_{\hat{z}}\right) + b_j m_2\left(\mu_{\hat{z}}, V_{\hat{z}}\right) + c_j m_1\left(\mu_{\hat{z}}, V_{\hat{z}}\right) \\
\mathrm{E}\left[w_j\left(\hat{z}\right)\right] &= a_j m_2\left(\mu_{\hat{z}}, V_{\hat{z}}\right) + b_j m_1\left(\mu_{\hat{z}}, V_{\hat{z}}\right) + c_j
\end{aligned}$$

and $m_t\left(\mu, V\right)$ is the t^{th} (non-central) moment of a univariate Gaussian distribution with mean μ and variance V.

3.4.3 Characterising distributions

In order to carry out any of the calculations in the approximate backward induction procedure, we need to be able to characterise the distributions $p\left(q|z_{[j]}, w_{[j]}, d_{[j]}\right)$ and $p\left(z_{[j]}|w_{[j]}, d_{[j]}\right)$ for any stage j of the algorithm. For simple, conjugate model specifications (such as the Gaussian-Gaussian specification used in Section 3.4.2), or for relatively simple, non-conjugate specifications, where the conditional and posterior distributions can be evaluated relatively easily, this does not present an issue; in these situations, we can just run the algorithm (Section 3.4.1, algorithm 2) using the these distributions. For general models, however, the relationship $p\left(z_{[j]}|q, w_{[j]}, d_{[j]}\right)$ has the potential to be non-linear, complex and slow to evaluate; in these situations, we can turn to the Bayesian uncertainty analysis framework presented in Sections 2.3.1 and 2.4.2 for assistance.

We assume that our data z_j are noise-corrupted measurements of some underlying system, as in Section 2.3.1

$$z_{jk} = y_k\left(w_j, d_j\right) + \epsilon_k\left(w_j, d_j\right)$$

where independence between $y\left(.\right)$ and $\epsilon\left(.\right)$ is assumed. We also assume that we have a simulator $f\left(.\right)$ which gives an imperfect representation of the system, relating the two together by including the discrepancy between the simulator and the system

$$y_k\left(w_j, d_j\right) = f_k\left(q, w_j, d_j\right) + \delta_k\left(w_j, d_j\right)$$

where, as in Section 2.3.1, we assume that δ is independent of $\{f, q\}$. In this context, we assume that q is the 'best input' setting for our simulator; we assume that running

$f(.)$ at setting q provides all of the information available from $f(.)$ about the system $y(.)$. In the design problem, we are interested in using the data z_j to learn about this setting.

We now approximate the complex forward relationship between general parameter setting t and the model output through the common emulator specification

$$f_k(t, w_j, d_j) = \sum_p \beta_{kp} g_p(t, w_j, d_j) + r_k(t, w_j, d_j) .$$

We make a second-order prior specification for both β and r, and we observe a set of runs F on f at known locations, before using these to obtain expressions for the adjusted predictions $\mathrm{E}_F[f(.)]$ and $\mathrm{Cov}_F[f(.), f(.')]$ at any new input setting (Section 2.2.2).

Propagating uncertainty If we take care when creating this approximation to f, we can ensure that we may carry out the uncertainty propagation calculations in Section 2.4.2 with relatively little computational effort, allowing us to obtain moments of $\hat{f}(w_j, d_j) = f(q, w_j, d_j)$ by computing expectations over $p(q)$.

In turn, these moments allow us to compute the moments of the system $y(.)$ by incorporating beliefs about the discrepancy $\delta(.)$, and then the data z_j by incorporating beliefs about the error structure $\epsilon(.)$; since the data are all assumed to be generated under the same model structure, we can use the specification in Section 2.2.2 and the calculations in Section 2.4.2 to compute the expectation $\mathrm{E}[z_j|w_j, d_j]$ of the data at any stage of the problem and the covariance $\mathrm{Cov}[z_j, z_k|w_j, d_j, w_k, d_k]$ between the data sets at any two stages. Having computed this full joint specification for all of the data, we can perform another adjustment to find the moments of any individual data set z_j given all previously-observed data $z_{[j-1]}$; re-introducing the summation convention, and stipulating that indices in square brackets are not summed over, we have

$$\begin{aligned}\mathrm{E}_{z_{[j-1]}}\left[z_{jk}|w_{[j]}, d_{[j]}\right] &= \mathrm{E}\left[z_{jk}|w_j, d_j\right] + \mathrm{Cov}\left[z_{jk}, z_{pq}|w_{[j]}, d_{[j]}\right] \\ &\quad \times \mathrm{Var}\left[z_{[j-1]}|w_{[j-1]}, d_{[j-1]}\right]^{-1}_{pqrs}\Big[z_{rs} - \mathrm{E}\left[z_{rs}|w_{[j-1]}, d_{[j-1]}\right]\Big]\end{aligned}$$

$$\begin{aligned}\mathrm{Var}_{z_{[j-1]}}\left[z_{jk}, z_{jl}|w_{[j]}, d_{[j]}\right] &= \mathrm{Var}\left[z_{jk}|w_j, d_j\right] - \mathrm{Cov}\left[z_{jk}, z_{pq}|w_{[j]}, d_{[j]}\right] \\ &\quad \times \mathrm{Var}\left[z_{[j-1]}|w_{[j-1]}, d_{[j-1]}\right]^{-1}_{pqrs} \mathrm{Cov}\left[z_{rs}, z_{jl}|w_{[j]}, d_{[j]}\right]\end{aligned}$$

where the sums over p and r range from $1, \ldots, (j-1)$, the sums over q and s range over $1, \ldots, n_{z_p}$ and $1, \ldots, n_{z_r}$ respectively, and the elements of $\mathrm{Var}\left[z_{[j-1]}|w_{[j-1]}, d_{[j-1]}\right]^{-1}$ are the elements of the inverse of

$$\mathrm{Var}\left[z_{[j-1]}|w_{[j-1]}, d_{[j-1]}\right]_{pqrs} = \mathrm{Cov}\left[z_{pq}, z_{rs}|w_{[j-1]}, d_{[j-1]}\right]$$

re-shaped into an appropriate four-dimensional array.

We use these adjusted moments for the data to characterise the conditional distributions $p\left(z_j|z_{[j-1]}, w_{[j]}, d_{[j]}\right)$ wherever they are encountered, as an approximation to the true conditional distribution, which is too expensive for us to work with; we may choose the distribution $p(.)$ that we feel best represents the behaviour of z_j, and then specify its parameters according to these moments. Often, the desire to ensure computational simplicity will come into play here; if we choose, for example, a Gaussian or a uniform distribution, then this will have the potential to simplify some of the numerical calculations that we need to perform later (Section 3.4.6).

Learning about q We can also use our emulator to approximately characterise the distributions $p\left(q|z_{[j]}, w_{[j]}, d_{[j]}\right)$ needed to evaluate the risks (3.3.6); the calculations described in Section 2.4.3 can be carried out in order to obtain $\mathrm{Cov}\left[q_l, z_{jk}|w_j, d_j\right]$ for any individual element of the dataset at stage j. We can then carry out an adjustment upon learning $z_{[j]}$ to obtain

$$\begin{aligned}\mathrm{E}_{z_{[j]}}\left[q_l|w_{[j]}, d_{[j]}\right] &= \mathrm{E}\left[q_l\right] + \mathrm{Cov}\left[q_l, z_{pq}|w_{[j]}, d_{[j]}\right] \\ &\quad \times \mathrm{Var}\left[z_{[j]}|w_{[j]}, d_{[j]}\right]^{-1}_{pqrs}\left[z_{rs} - \mathrm{E}\left[z_{rs}|w_{[j]}, d_{[j]}\right]\right]\end{aligned}$$

$$\begin{aligned}\mathrm{Cov}_{z_{[j]}}\left[q_l, q_m|w_{[j]}, d_{[j]}\right] &= \mathrm{Cov}\left[q_l, q_m\right] - \mathrm{Cov}\left[q_l, z_{pq}|w_{[j]}, d_{[j]}\right] \\ &\quad \times \mathrm{Var}\left[z_{[j]}|w_{[j]}, d_{[j]}\right]^{-1}_{pqrs} \mathrm{Cov}\left[z_{rs}, q_m|w_{[j]}, d_{[j]}\right]\end{aligned}$$

where the sums over p and r range from $1, \ldots, j$, and the sums over q and s range over $1, \ldots, n_{z_p}$ and $1, \ldots, n_{z_r}$ respectively. Again, since we have used a Bayes linear

calculation to approximate a fully probabilistic one, we choose a suitable form for the distribution $p\left(q|z_{[j]}, w_{[j]}, d_{[j]}\right)$ and parametrize it by approximating the true conditional moments using the adjusted moments computed above. For some choices of loss function, however, this may not be necessary; if the loss results in a risk which is a function of only first- and second-order moments, then we can simply proceed using only the adjusted second-order specification that we have derived (see the discussion in Section 3.1.2).

3.4.4 Fitting a risk emulator

We fit the emulator (3.4.12) as described in Section 2.2.2. We first make a prior specification for the moments of the basis parameters, $\mathrm{E}\left[\alpha_j^{(i)}\right]$ and $\mathrm{Var}\left[\alpha_j^{(i)}\right]$, and for the covariance structure of the residual process, $\mathrm{Cov}\left[u_j^{(i)}(.), u_j^{(i)}(.')\right]$; then, we generate data from the risk function at this stage at known settings of the risk inputs, and use this data to adjust our prior moments. This allows us to make fast predictions for the risk at any new setting of its inputs with an uncertainty level which describes our confidence in these predictions. We consider some aspects of this model fit.

Generating risk evaluations: In order to fit the emulator, we generate evaluations of the risk at known input settings. At wave i, we denote the set of $N_j^{(i)}$ risk values that we use for the adjustment by $R_j^{(i)} = \{R_{j1}^{(i)}, R_{j2}^{(i)}, \ldots, R_{jN_j^{(i)}}^{(i)}\}$; $R_{jk}^{(i)}$ is the k^{th} evaluation of the risk, obtained at input setting $\{z_{[j]k}, w_{[j]k}, d_{[j]k}\}$. When generating the risk evaluations, there are two separate cases to consider:

- At the final stage, $j = n$, the data is generated directly from the risk (3.3.6) from an optimal terminal decision

$$R_{nk}^{(i)} = \rho_n^t\left[z_{[n]k}, w_{[n]k}, d_{[n]k}\right]$$

the amount of data that we can use to update the emulator is therefore limited by the computational effort required to evaluate (3.3.6), and the computational resources that we have available. In the case where this risk is closed-form (see Section 3.1.1) and the model is simple, generating data incurs almost no cost,

and the process of inverting the data covariance to update the emulator is the more rate-limiting step; however, if the model is more complex, or if there is no closed-form expression for the risk, then we must use numerical methods, making the data generation more costly. Where the terminal risk must be evaluated numerically, we may do this using an MCMC algorithm, or using the Bayesian integration technique presented in Section 2.4.1; any uncertainty about the terminal risk value resulting from its numerical evaluation is accounted for as measurement error when fitting the risk emulator.

- At any other stage $j = (n-1), \ldots, 1$, the risk is computed through comparison of the risk from an optimal terminal decision at the current stage with the risk from an optimally-designed experiment at the next stage

$$R_{jk}^{(i)} = \min \left[\rho_j^t \left[z_{[j]k}, w_{[j]k}, d_{[j]k} \right], s_{j+1}^{(i)} \left[z_{[j]k}, w_{[j]k}, d_{[j]k} \right] \right]. \tag{3.4.16}$$

  Our numerical approximations to the risks introduce uncertainty, and so $s_{j+1}^{(i)}$ is unknown. As discussed in Section 3.4.8, our emulator for the risk at stage $(j+1)$ induces an uncertainty specification for this quantity; we can compute the expectation $\mathrm{E}\left[s_{j+1}^{(i)} \left[z_{[j]k}, w_{[j]k}, d_{[j]k} \right] \right]$ for each input setting, and the covariances $\mathrm{Cov}\left[s_{j+1}^{(i)} \left[z_{[j]k}, w_{[j]k}, d_{[j]k} \right], s_{j+1}^{(i)} \left[z_{[j]l}, w_{[j]l}, d_{[j]l} \right] \right]$ between risks at each pair of inputs. We then use this uncertainty specification to characterise a multivariate Gaussian distribution, and we use samples drawn from this distribution to compute expectations $\mathrm{E}\left[R_{jk}^{(i)} \right]$ and covariances $\mathrm{Cov}\left[R_{jk}^{(i)}, R_{jl}^{(i)} \right]$ for the risk values. We fit our emulator to the expectations $\mathrm{E}\left[R_{jk}^{(i)} \right]$, using the covariances to characterise the measurement error structure.

Modelling choices: When building our model for the risk at stage j, we want to choose the basis functions $h_{jp}^{(i)}(.)$ and the covariance of the residual process $u_j^{(i)}(.)$ in such a way that we can obtain an accurate representation of the risk at any given input point; however, we also want to make sure that we can carry out the integrals (3.4.13) as cheaply as possible, and that the sampling procedure used to characterise the minimum of the risk (Section 3.4.8) does not become too computationally expensive. The following specifications may be suitable in a wide range of problems:

- **Risk as a mean function:** where we can compute the risk $\rho_j^t[.]$ from an optimal terminal decision directly, or relatively inexpensively using numerical methods, it is often the case that using this as a basis function for the regression term will account for a large amount of the systematic variation in the true risk; in parts of the input space where it is optimal to make an immediate decision, this basis function will completely account for the risk behaviour, and in parts of the space where we will continue sampling, the variation can be absorbed using other mean functions and the residual process.

  Where the expectation of $\rho_j^t[.]$ over z_j or w_j cannot be computed easily, as required in equation 3.4.13, then using $\rho_j^t[.]$ as a basis function can introduce complications at this stage of the approximation procedure; to get around this, we can instead use $\rho_j^t\left[\tilde{z}_{[j]}, \tilde{w}_{[j]}, d_{[j]}\right]$ as a basis function, where $\tilde{z}_{[j]} = \{z_1, \ldots, \mathrm{E}\left[z_j | z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\right]\}$ and $\tilde{w}_{[j]} = \{w_1, \ldots, \mathrm{E}\left[w_j | w_{[j-1]}, d_{[j]}\right]\}$ are the data and external input sets with the value at the current stage j replaced by the expectation conditional on the values from previous stages (for complex models where we use an approximating emulator, as in Section 3.4.3, we approximate the full conditional moments using the adjusted moments $\mathrm{E}_{z_{[j-1]}}\left[z_j | w_{[j-1]}, d_{[j-1]}\right]$ and $\mathrm{E}_{w_{[j-1]}}\left[w_j | d_{[j]}\right]$). The difference between this basis function and the true risk can then be explained using additional basis functions, the residual process or the nugget term. The terminal risk is used as part of the basis function for the emulator in both the simple example in Section 3.4.5 and the more complex one presented in Section 4.1.

- **Input space reduction:** in a problem with many input variables or large amounts of data, it may be the case that the majority of the variability in the risk function is driven by a small number of linear combinations of the $\{z_{[j]}, w_{[j]}, d_{[j]}\}$; in this instance, modelling the risk in terms of only these linear combinations has the potential to significantly reduce the complexity of the emulator that we fit, and the subsequent calculations. For example, in the case where we collect a large amount of data at each stage, we may choose to fit an emulator in terms of the sample mean of the z_j, or we might choose to identify the directions of canonical correlation between z_j and $R_j^{(i)}$ and model in terms

of the linear combinations of z_j that explain the most variability. Mardia et al. [1979] (chapter 11) provide an introduction to canonical correlation analysis.

- **Variable removal:** where there is little evidence that a variable has a systematic effect on the value of the risk, it may be appropriate to remove it from the model entirely, absorbing any remaining variability using an uncorrelated residual term; for example, where all distributions are Gaussian, and the loss function is an un-weighted quadratic, the data z_j has no effect on the risk, and so can safely be removed from the model without decreasing our ability to explain the behaviour of the risk.

Specifying the prior: Often, our prior knowledge about the behaviour of the risk will be poor, and so specifying an appropriate basis function set and corresponding prior coefficient moments is challenging; the amount of risk data that we have available, however, is only limited by the amount of computer time that we can devote to generating it. In most cases, therefore, we can generate an additional, smaller set of risk evaluations which can be used to carry out an initial linear regression. This regression can be used to fix the prior moments $\mathrm{E}\left[\alpha_{jp}^{(i)}\right]$ and $\mathrm{Cov}\left[\alpha_{jp}^{(i)}, \alpha_{jq}^{(i)}\right]$ of the basis coefficients. We can also use the residuals of this regression fit to empirically fix the prior marginal covariance $\mathrm{Var}\left[u_j^{(i)}(.)\right]$ of the residual process.

Determining the nugget variance: Where we include a nugget term $\xi_j^{(i)}$, we do so to account for variability in the risk which cannot be explained using the regression and residual terms which we have selected; for example, where we have specified that the systematic components depend only on a low-dimensional summary of the observed data, and we must account for the risk variability which cannot be explained using this summary. In most cases, we can assess the level of this variability by generating further samples from the risk.

Where we can, we assess $\mathrm{Var}\left[\xi_j^{(i)}\right]$ by holding constant those inputs for which we expect the regression and residual components to explain risk variability, and then varying the inputs which will induce the variability that we want to capture using the nugget; this should be done at a number of different settings of the fixed inputs,

to check that the level of variability does not change drastically across the input space. Var $\left[\xi_j^{(i)}\right]$ is fixed to the variance of the sample (or the average variance of the samples from different settings of the fixed inputs).

Fitting the emulator: For a particular setting of the correlation parameters, we can adjust our beliefs about $r_j^{(i)}$ using the calculations outlined in Section 2.2.2; our adjusted expectation $\mathrm{E}_{R_j^{(i)}}\left[r_j^{(i)}[.]\right]$ is computed as in equation (2.2.12) and our adjusted covariance $\mathrm{Cov}_{R_j^{(i)}}\left[r_j^{(i)}[.], r_j^{(i)}[.']\right]$ between function values at different input settings is computed as in equation (2.2.13) .

Determining correlation parameters: Having used the initial regression and residual analysis to empirically fix the regression priors and the marginal variance of $u_j^{(i)}(.)$, it remains to determine any parameters which govern the degree of correlation between input settings for the residual; since the dependence of the covariance function on these is typically non-linear, we cannot perform the usual Bayes linear analysis by specifying a prior and using the data to adjust. Instead, we fix the correlation parameters using a leave-one-out cross validation procedure [Rasmussen and Williams, 2006]; we leave out each point in turn, and predict it using the fit to the remainder. To assess the quality of the fit for each candidate correlation parameter setting, we use the sum of the predictive Gaussian likelihoods for all points. For further discussion of cross validation, see Section 2.2.3.

Checking inter-wave correspondence: Under the procedure 2, we have the option of performing multiple waves of analysis. At the first wave ($i = 1$), we emulate the risk function at each stage for the first time; then, at subsequent waves ($i = 2, 3, \dots$), we repeatedly re-emulate the same risk functions over sub-regions of the design input space. We hope that as we focus our models on sub-regions of the design input space, we will be able to model risk behaviour more accurately; at a minimum however, we expect that the predictive error bars of the emulator fitted at wave i should overlap with those of the emulators fitted at stages $1, \dots, (i-1)$. This property can be checked to ensure that our models are behaving as they should. If we do not see overlap between emulators at consecutive waves, this could be an

indication of problems with our model specification; for example, failure to fully acknowledge the level of our uncertainty about the risk at one or more stages.

3.4.5 Illustrative example: fitting

We now illustrate the risk emulator fitting procedure outlined in Section 3.4.4 through application to the problem outlined in Section 3.4.2; we consider an experimental design problem in which we may make at most $n = 2$ observations, and where we must eventually decide whether it is worth carrying out the first experiment, or whether we should just make an immediate decision under the prior.
We set up the elements of the problem as follows:

- the vector of basis functions at each stage is $g(d_j) = (1, (1 + d_j)(1 - d_j))^{\mathrm{T}}$;
- the design parameter at each stage is constrained to lie in the range $d_j \in [-1, 1]$;
- the prior moments of the model parameters are set to $\mu_q = (0, 0)^{\mathrm{T}}$, $V_q = \mathrm{Diag}((0.5^2, 0.5^2))$;
- the measurement precisions are set to be $\lambda_{\epsilon_1} = 1/(0.5^2)$, $\lambda_{\epsilon_2} = 1/(0.1^2)$, so we have the option of a more accurate measurement at the second stage;
- our losses depend on the data value $\hat{z}$ at new input setting $\hat{d} = 1/\sqrt{2}$;
- the weightings for the loss function are set to be the same at all stages, with $a_j = c_j = 1$ and $b_j = 0$;
- the measurement costs are set to be constant across the design space, and so that the first observation is cheaper than the second; $c_1(d_1) = 0.05$ and $c_2(d_2) = 0.2$.

Having specified all of these elements, we are ready to start our analysis of the risk.

For this example, we adopt the strategy (discussed in Section 3.4.4) of using the modified terminal risk as a mean function; with reference to the general emulator

form (3.4.12), we specify that $h_{21}^{(1)} = 1$

$$h_{22}^{(1)}\left(z_{[2]}, d_{[2]}\right) = \rho_2^t\left[\tilde{z}_{[2]}, d_{[2]}\right]$$

where $\tilde{z}_{[2]} = (z_1, \mathrm{E}\left[z_2|z_1\right])^{\mathrm{T}}$, so z_2 has been replaced in the risk function by its expectation conditional on z_1. This modified risk function is useful as a basis term, since it closely mirrors the behaviour of the risk $\rho\left[z_{[2]}, d_{[2]}\right]$, but does not have a direct dependence on z_2, which means that it will be simple to integrate when we need to compute expectations of the risk in Sections 3.4.6 and 3.4.7.

For the residual, we choose a simple, separable squared exponential covariance function

$$\mathrm{Cov}\left[u_2^{(1)}\left(z_{[2]}, d_{[2]}\right), u_2^{(1)}\left(z'_{[2]}, d'_{[2]}\right)\right] = v_2^{(1)} \prod_{k=1}^{2}\left[c\left(d_k, d'_k|\lambda_d\right) c\left(z_k, z'_k|\lambda_z\right)\right]$$

where $v_2^{(1)} = \mathrm{Var}\left[u_2^{(1)}\right]$ is the marginal variance parameter and $c\left(.,.|\lambda\right)$ is a squared exponential correlation function with correlation parameter for a scalar input

$$c\left(\theta, \theta'|\lambda\right) = \exp\left[-\frac{\lambda}{2}(\theta - \theta')^2\right].$$

For further details relating to the SE correlation function, see Appendix B.1.

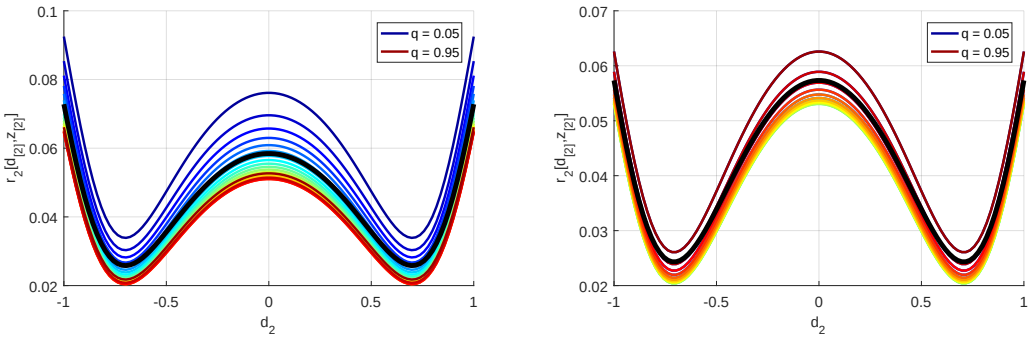
Since this is the final stage of the problem, $\rho_2\left[.\right] = \rho_2^t\left[.\right]$, and the sole basis function is this terminal risk evaluated at a central setting of the observation z_2; the purpose of the residual process in this instance, therefore, is to absorb structured variability in the risk function due to z_2 which is not captured by the regression component (over the range of possible inputs $\{z_{[1]}, d_{[2]}\}$). The squared exponential is chosen because it is simple to integrate with respect to a Gaussian distribution, and so will ensure that the calculations that we must perform in the coming sections (3.4.6 and 3.4.7) are relatively simple. The correlation parameters $\{\lambda_d, \lambda_z\}$ are fixed to be the same at both stages $k = 1, 2$, in order to simplify the problem of determining them from the data.

Having selected our mean and covariance functions, we are in a position to start fitting the risk model. First, we must specify our prior beliefs about the regression coefficients $\{\alpha_{21}^{(1)}\alpha_{22}^{(1)}\}$. As discussed in Section 3.4.4, we do this by performing an

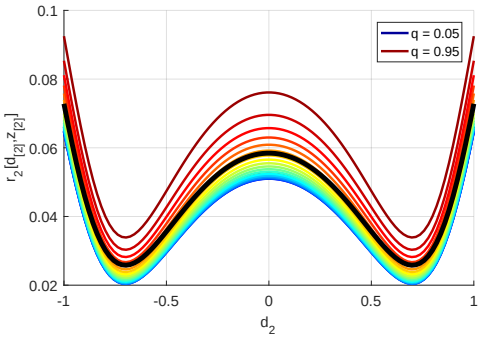
initial linear regression, using a set of 200 risk evaluations (with the d_j where we evaluate the risk chosen according to a Latin hypercube). We fix the prior moments to the parameter estimates obtained using this regression fit, giving $\mathrm{E}\left[\alpha_{21}^{(1)}\right] = 0.0037$ and $\mathrm{E}\left[\alpha_{22}^{(1)}\right] = 1.01$, with $\mathrm{Var}\left[\alpha_{21}^{(1)}\right] = (2.4 \times 10^{-7})^2$ and $\mathrm{Var}\left[\alpha_{22}^{(1)}\right] = (5.59 \times 10^{-6})$. Next, we fix the marginal variance of the residual process using the residuals from this regression fit (again, as discussed in Section 3.4.4); this results in us specifying $\mathrm{Var}\left[u_2^{(1)}(.)\right] = v_2^{(1)} = (0.0101)^2$. In this instance, we believe that all of the variability in $\rho_2[.]$ will be accounted for by the regression and residual components, and so we simply set $\mathrm{Var}\left[\xi_2^{(1)}\right] = (10^{-3})^2$ in order to ensure numerical stability of the update calculations.

We fix the correlation lengths for the covariance function using leave-one-out cross-validation. First, we generate $R_2^{(1)}$, the set of $N_2^{(1)} = 300$ risk evaluations that we will use to perform the joint update for the regression and residual components; we also predict $\mathrm{E}\left[R_{2k}^{(1)}\right]$ for each risk evaluation using the prior specification for the $\alpha_2^{(1)}$ computed above, and compute the residuals $D_k = R_{2k}^{(1)} - \mathrm{E}\left[R_{2k}^{(1)}\right]$. We generate a Latin hypercube of 2000 points $\{\lambda_d, \lambda_z\}$; then, we test the quality of each such specification by leaving out each of the D_k out in turn, fitting the model to the remaining residuals, and computing the predictive log-Gaussian likelihood for D_k under this fit. We fix the correlation parameters to the setting which minimises the sum of these likelihoods. See Section 2.2.3 for further details of this procedure.

The fitting procedure for this emulator is illustrated in Figures 3.2 and 3.3. Figure 3.2 shows the relationship between the expected regression surface $\sum_p \mathrm{E}\left[\alpha_{2p}^{(1)}\right] h_{2p}^{(1)}(.)$ and the risk $\rho_2[.]$, for different settings of $\{z_{[2]}, d_{[2]}\}$; we see that, in all of the displayed cases, the regression surface mirrors the behaviour of the risk function across the design space, with systematic deviations from this level driven by the value of the observed data z_2. Figure 3.3 shows the final fitted emulator, after the marginal variance and the correlation parameters of the residual process have been determined as described above.



(a) (b)



(c)

Figure 3.2: Plots showing the difference between the mean regression surface and the true risk as a function of d_2 for different settings of the parameters $\{z_{[1]}, d_{[1]}\}$; in Figure 3.2(a), we set $d_1 = -0.5$ and z_1 such that $P(z_1|d_1) = 0.25$, with the colour scale signifying different values $P(z_2|d_2) = q_{z_2}$ at which we generate z_2; in Figure 3.2(b), the same is plotted by for $d_1 = 0$ and $P(z_1|d_1) = 0.5$; lastly, Figure 3.2(c) has $d_1 = 0.5$ and $P(z_1|d_1) = 0.75$. In all cases, the surface $\sum_p \alpha_{2p}^{(1)} h_{2p}^{(1)}$ is shown as a black line. In each case, the mean regression surface tracks the shape of the risk function, with the difference between this surface and the true risk clearly a systematic function of d_2 and z_2.

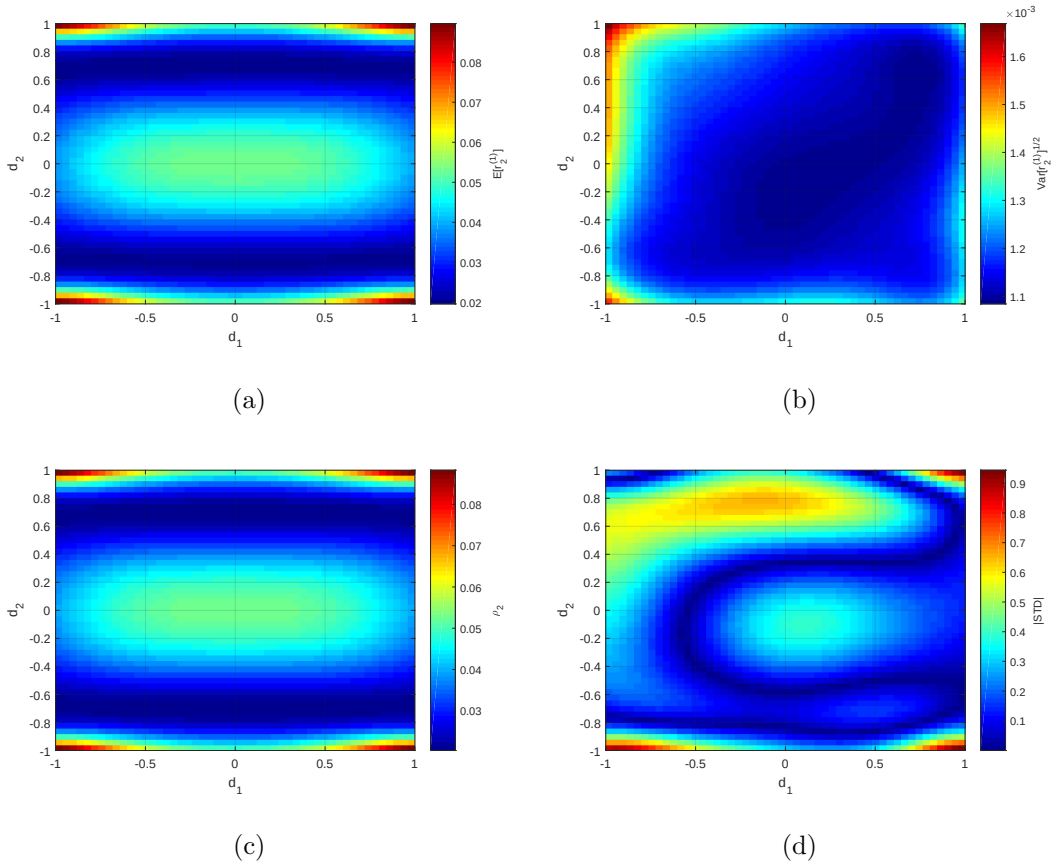


Figure 3.3: Moments of the emulator $r_2^{(1)}[.]$, fitted in Section 3.4.5: Figure 3.3(a) shows the adjusted expectation $\mathrm{E}_{R_2^{(1)}}\left[r_2^{(1)}\right]$ as a function of d_1 and d_2, with the observations $z_{[2]}$ fixed to the mean values of their distribution at each point; Figure 3.3(b) shows the corresponding adjusted standard deviation $\mathrm{Var}_{R_2^{(1)}}\left[r_2^{(1)}\right]^{1/2}$; Figure 3.3(c) shows the true value of the risk ρ_2^t, and Figure 3.3(d) shows the absolute value of the standardised distance $(\rho_2^t - \mathrm{E}_{R_2^{(1)}}\left[r_2^{(1)}\right])/\mathrm{Var}_{R_2^{(1)}}\left[r_2^{(1)}\right]^{1/2}$ between the prediction and the truth at each point.

3.4.6 Computing expectations

We use the model $r_j^{(i)}$ fitted in Section 3.4.4 to characterise an approximation $\bar{r}_j^{(i)}$ to the expected risk $\bar{\rho}_j$ (defined in equation (3.3.10)). As outlined in Section 2.4.1, the moments of the expectation of the process can be derived by integrating the moments of the process directly. The expectation of $\bar{r}_j^{(i)}$ is

$$\mathrm{E}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}\right]\right] = \int \mathrm{E}_{R_j^{(i)}}\left[r_j^{(i)}\left[.\right]\right] p\left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]}\right) p\left(w_j | w_{[j-1]}, d_{[j]}\right) dz_j dw_j$$

and the covariance between $\bar{r}_j^{(i)}$ values at any pair of input settings is

$$\mathrm{Cov}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}\right], \bar{r}_j^{(i)}\left[z'_{[j-1]}, w'_{[j-1]}, d'_{[j]}\right]\right] = \int \mathrm{Cov}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[.\right], \bar{r}_j^{(i)}\left[.'\right]\right] p\left(z_j, w_j | z_{[j-1]}, w_{[j-1]}, d_{[j]}\right) \times p\left(z'_j, w'_j | z'_{[j-1]}, w'_{[j-1]}, d'_{[j]}\right) dz_j dw_j dz'_j dw'_j \ .$$

These definitions correspond to those in equations (2.4.19) and (2.4.21) from Section 2.4.1.

The ease with which we can compute these moments depends on how we have constructed the emulator: if we took care to choose basis and covariance functions which can be integrated analytically with respect to $p\left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]}\right)$ and $p\left(w_j | w_{[j-1]}, d_{[j]}\right)$, then the procedure for generating a prediction from the integrated emulator requires no greater computational expense than the process of generating a prediction from the emulator itself. However, if choosing basis and covariance functions which can be integrated analytically against these distributions is not possible without sacrificing too much of the explanatory power of the emulator then we must resort to numerical integration in order to compute these terms. As always, the predictive power of the emulator must be traded off against the speed with which it (and its integrated moments) can be evaluated.

3.4.7 Illustrative example: expectation

Having fitted the risk emulator in Section 3.4.5, we must now use this to compute our beliefs about the expected risk, as outlined in Section 3.4.6; because of our

choices of basis and covariance function, this is a relatively simple job. Since there is no external parameter in our illustrative problem, we must only compute the expectation of the risk emulator over $p\left(z_2|z_{[1]}, d_{[2]}\right)$. Because of the simple model form, this distribution is a Gaussian, the form of which is given in equation 3.4.15. Since the basis functions have no dependence on z_2, these integrals are easy; we have that

$$\bar{h}_{2p}^{(1)}\left(z_{[1]}, d_{[2]}\right) = h_{2p}^{(1)}\left(z_{[1]}, d_{[2]}\right)$$

for both functions $p = 1, 2$. For the covariance function, the integral is relatively simple to compute, because of the choice of a separable squared exponential form; integrating once, we have that

$$\begin{aligned}\text{Cov}\left[\bar{u}_2^{(1)}\left(z_{[1]}, d_{[2]}\right), u_2^{(1)}\left(z'_{[2]}, d'_{[2]}\right)\right] &= v_2^{(1)}\Big[\prod_{k=1}^{2} c\left(d_k, d'_k|\lambda_d\right)\Big] c\left(z_1, z'_1|\lambda_z\right) \\ &\quad \times \bar{c}\left(\mu_{z_2}\left(z_{[1]}, d_{[2]}\right), V_{z_2}\left(d_{[2]}\right), z'_2|\lambda_z\right)\end{aligned}$$

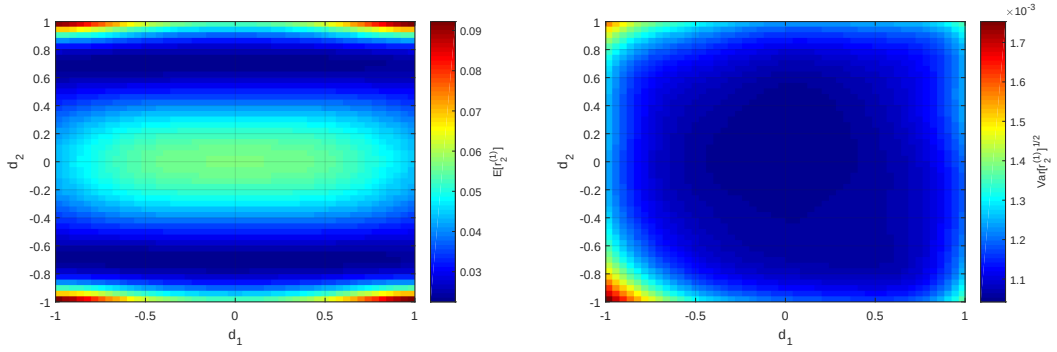
where $\bar{c}\left(.,.|\lambda_z\right)$ is computed by integrating the covariance function $c\left(.,.|\lambda_z\right)$ with respect to the distribution $p\left(z_2|\mu_{z_2}\left(z_{[1]}, d_{[2]}\right), V_{z_2}\left(d_{[2]}\right)\right)$; details of this calculation are provided in the Appendix B.1.2. Integrating a second time, this time with respect to the distribution $p\left(z'_2|\mu_{z_2}\left(z'_{[1]}, d'_{[2]}\right), V_{z_2}\left(d'_{[2]}\right)\right)$, we find

$$\begin{aligned}\text{Cov}\left[\bar{u}_2^{(1)}\left(z_{[1]}, d_{[2]}\right), \bar{u}_2^{(1)}\left(z'_{[1]}, d'_{[2]}\right)\right] &= v_2^{(1)}\Big[\prod_{k=1}^{2} c\left(d_k, d'_k|\lambda_d\right)\Big] c\left(z_1, z'_1|\lambda_z\right) \\ &\quad \times \bar{\bar{c}}\left(\mu_{z_2}\left(z_{[1]}, d_{[2]}\right), V_{z_2}\left(d_{[2]}\right), \mu_{z_2}\left(z'_{[1]}, d'_{[2]}\right), V_{z_2}\left(d'_{[2]}\right)|\lambda_z\right)\end{aligned}$$

where again, $\bar{\bar{c}}\left(.,.\right)$ is computed for a general Gaussian distribution in Appendix B.1.2. Plugging these quantities into the expressions (2.4.19) and (2.4.21) allows us to evaluate the moments $\text{E}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\left[.\right]\right]$ and $\text{Cov}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\left[.\right], \bar{r}_2^{(1)}\left[.'\right]\right]$ of the emulator for the expected risk. Figure 3.4 plots the moments of $\bar{r}_2^{(1)}$ for a range of different settings of (d_1, d_2).

3.4.8 Characterising the minimum risk

To run the procedure outlined in Section 3.4.1, we must be able to assess which design inputs could plausibly be optimal at any given stage, and be able to characterise our corresponding uncertainty about the value of the risk at the minimum.



(a) (b)

Figure 3.4: Plots of the emulator $\bar{r}_2^{(1)}$ for the expected risk. Figure 3.4(a) shows the expectation $\mathrm{E}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\right]$ for a range of different settings of (d_1, d_2), and Figure 3.4(b) shows the standard deviations $\mathrm{Var}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\right]$ corresponding to the same points. For all predictions, z_1 is fixed so that $P\left(z_1|d_1\right) = 0.5$.

Characterising exactly the distributions of the extrema of a stochastic function is a complex and open problem (see, for example, Henning and Schuler [2012] or Adler [1990] for discussions), and so we do not attempt to do this; instead, we adopt an approximate procedure, in which we sample the risk at a space-filling set of inputs and find the design which minimises the risk over this design, checking that the number of points which we generate does not unduly influence the size of the resulting space.

Candidate design space We represent our uncertainty about the optimal design d_j^* by using the emulator $\bar{r}_j^{(i)}$ to define a 'candidate design space' $\mathcal{D}_j^{(i)}$. For a given setting of the inputs $\{z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\}$, we generate a space filling set of $M_j^{(i)}$ trial design inputs $\{d_{j1}, d_{j2}, \ldots, d_{jM_j^{(i)}}\}$ inside the space $\mathcal{D}_j^{(i-1)}$ (where $\mathcal{D}_j^{(0)} = \mathcal{D}_j$, the full design space), and we sample corresponding risks $\bar{r}_j^{(i)}$ from a multivariate Gaussian characterised by the moments $\mathrm{E}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[.\right]\right]$ and $\mathrm{Cov}\left[\bar{r}_j^{(i)}\left[.\right], \bar{r}_j^{(i)}\left[.'\right]\right]$. We accept as a 'candidate design' $\tilde{d}_j$ the trial design input which minimises the risk over this set. The candidate design space $\mathcal{D}_j^{(i)}$ is simply the set of design inputs which are identified as candidate designs by this procedure.

The sampling procedure that we use to identify candidate designs is summarised in

algorithm 3. As defined, this procedure is recursive; we must be able to generate designs from the space $\mathcal{D}_j^{(i-1)}$ in order to be able to generate designs from the space $\mathcal{D}_j^{(i)}$. The computational complexity of running algorithm 3 will therefore grow rapidly as we progress through the waves. If running the full procedure for a given problem is computationally challenging, and the candidate designs that we generate lie within identifiable sub-regions of the full design space, then we will generally choose to approximate the full procedure by characterising $\mathcal{D}_j^{(i-1)}$ using simple limits. This is the approach that we take in both the simple linear model example (Section 3.4.12) and the more complex example presented in Section 4.2.

Algorithm 3 Sample a candidate design $\tilde{d}_j$ at stage j, wave i.

1: Generate $M_j^{(i)}$ space-filling trial designs $\{d_{j1}, d_{j2}, \ldots, d_{jM_j^{(i)}}\}$ within the candidate space $\mathcal{D}_j^{(i-1)}$

2: Jointly sample $\bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, \{d_{[j-1]}, d_{jk}\}\right]$ values for the set of all trial designs $\{d_{jk}\}$ from a Gaussian distribution

3: Set

$$\tilde{d}_j = \arg\min_{d_{jk}} \left[\bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, \{d_{[j-1]}, d_{jk}\}\right] + c_j\left(d_{jk}\right)\right]$$

Moments of $s_j^{(i)}$ Having fixed our procedure for characterising the candidate design space, we use this procedure to characterise our uncertainty about the risk $s_j^{(i)}$ at the minimum (defined in (3.4.14)). We do not know the location of the true optimal design d_j^*, but our uncertainty about this quantity is characterised by the sampling procedure 3; we therefore compute moments of $s_j^{(i)}$ by substituting candidate designs $\tilde{d}_j$ for the true optimal design setting and computing the average behaviour of the risk over the design space by sampling.

Our expectation for $s_j^{(i)}$ is computed using the law of total expectation

$$\mathrm{E}\left[s_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\right]\right] = \mathrm{E}\left[\mathrm{E}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[\tilde{d}_j\right]\right] + c_j\left(\tilde{d}_j\right)\right] \tag{3.4.17}$$

and our covariance between $s_j^{(i)}$ values at different input settings is computed using the law of total covariance

$$\begin{aligned}\mathrm{Cov}\left[s_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\right], s_j^{(i)}\left[z'_{[j-1]}, w'_{[j-1]}, d'_{[j-1]}\right]\right] = \\ \mathrm{Cov}\left[\mathrm{E}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[\tilde{d}_j\right]\right] + c_j\left(\tilde{d}_j\right), \mathrm{E}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[\tilde{d}_j\right]\right] + c_j\left(\tilde{d}_j\right)\right] \\ + \mathrm{E}\left[\mathrm{Cov}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[\tilde{d}_j\right], \bar{r}_j^{(i)}\left[\tilde{d}_j\right]\right]\right] \quad (3.4.18)\end{aligned}$$

where in all cases, the outer expectations and covariances are computed with respect to $\tilde{d}_j$ and we have suppressed the dependencies on the inputs $\{z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\}$ and $\{z'_{[j-1]}, w'_{[j-1]}, d'_{[j-1]}\}$ on the right-hand side of all expressions.

For many problems, it will be computationally infeasible to assess the moments (3.4.17) and (3.4.18) for every input setting using multiple candidate design settings sampled according to algorithm 3; in such problems we opt to characterise both moments using only a single candidate design value $\tilde{d}_j$. In this case, we fix the expectation (3.4.17) to the value $\mathrm{E}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[\tilde{d}_j\right]\right]$, and we fix the second term in the covariance (3.4.18) to be the covariance $\mathrm{Cov}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[\tilde{d}_j\right], \bar{r}_j^{(i)}\left[\tilde{d}_j\right]\right]$ between $\bar{r}_j^{(i)}$ values at this candidate design setting. We approximate the first component of the covariance by assuming that it is fixed across the input space $\{z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\}$. We approximate the variance of the expectation $\mathrm{E}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[\tilde{d}_j\right]\right] + c_j\left(\tilde{d}_j\right)$ for any input setting by sampling multiple candidate designs at a number of different input settings, computing the variance of each sample, and fixing $\mathrm{Var}\left[\mathrm{E}_{R_j^{(i)}}\left[\bar{r}_j^{(i)}\left[\tilde{d}_j\right]\right] + c_j\left(\tilde{d}_j\right)\right]$ to the mean of these variances; we then fix all covariances between expectations at different input settings to be zero.

Sensitivity to trial design size When using this procedure, it is important to assess the sensitivity of the result to the size of the Latin hypercube used in the sampling procedure. Provided the trial design size $M_j^{(i)}$ is large enough, increasing the size of the trial design will not change the characteristics of the space being targeted by the algorithm 3; however, a sample which is too large wastes computational effort. The size of the Latin hypercube that we need to generate is affected by the dimensionality of the design space and our level of uncertainty about the risk function within it. Determining this optimal Latin hypercube size, beyond which

there is no change in the sampled minimum characteristics, is generally a challenging task.

Since the determination of the optimal sample size looks to be too challenging, we instead plot the characteristics of the candidate design space $\mathcal{D}_j^{(i)}$ for a number of different sample sizes $M_j^{(i)}$ and select a sample size which we feel represents an adequate trade-off between computation time and stability of these characteristics. If we double $M_j^{(i)}$ and there is little change in the characteristics of the sampled $\tilde{d}_j$ for a range of different $\{z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\}$, then we may choose to use the smaller value of $M_j^{(i)}$, or to investigate using an even smaller value than this. However, if doubling the sample size results in a noticeable change in the behaviour of the $\tilde{d}_j$ values, then we should use the larger sample size, or if this looks to be infeasible, investigate intermediate values for $M_j^{(i)}$.

3.4.9 Illustrative example: minimum sampling

We illustrate aspects of the procedure outlined in Section 3.4.8 through application to the linear model example. First, we carry out a rough assessment of the size $M_2^{(1)}$ of the Latin hypercube of design inputs that we use to interrogate the emulator $\bar{r}_2^{(1)}$ when running the algorithm 3. We do this by trying two different design sizes, 200 and 400, and comparing the characteristics of the resulting candidate design spaces. We find that doubling the design size to 400 has little noticeable effect on the range of values $\tilde{d}_2$ which we sample, and so we choose to fix $M_2^{(1)} = 200$, the smaller of the two values. In a larger problem, we might also look at the effect of choosing an even smaller trial design size; however, in this simple problem, it is not as critical that we make every possible computational saving, and so we do not examine this further. Figure 3.5 shows the expectations (3.4.17) and marginal variances (3.4.18) of $s_2^{(1)}\left[z_{[1]}, d_{[1]}\right]$ for a range of different values of z_1 for fixed $d_1 = 0$; the moments at each point are assessed by evaluating the outer expectations and covariances in the expressions (3.4.17) and (3.4.18) using 20 candidate designs generated according to the procedure 3.

When running the approximate backward induction procedure at stage 1, we will adopt the strategy outlined in Section 3.4.8 of using only a single candidate design

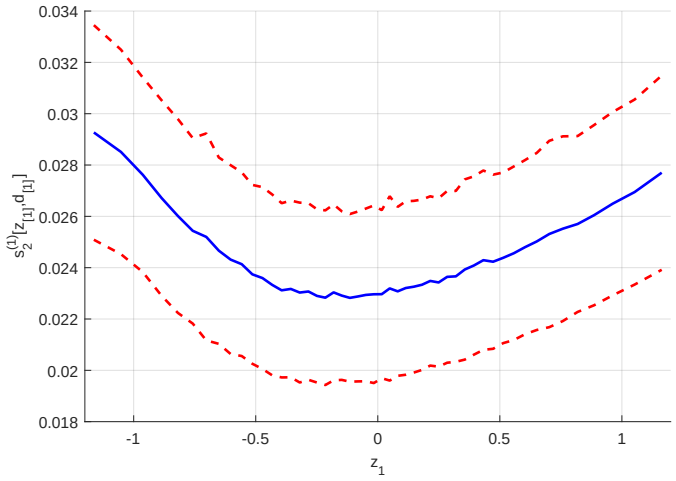


Figure 3.5: Plots of candidate minimum samples generated according to the procedure in Section 3.4.8 for the linear model example (Section 3.4.9): the first stage design is fixed to $d_1 = 0$, and z_1 is varied across the quantiles of the distribution $p\left(z_1|d_{[1]}\right)$; 20 minimum samples are generated for each setting $\{z_1, d_1\}$, and the resulting values of the moments (3.4.17) and (3.4.18) are shown. $\mathrm{E}\left[s_2^{(1)}\right]$ is shown in blue, and error bars $\mathrm{E}\left[s_2^{(1)}\right] \pm 3 \times \mathrm{Var}\left[s_2^{(1)}\right]^{1/2}$ are shown in dashed red.

sample to evaluate the moments (3.4.17) and (3.4.18) of $s_2^{(1)}$ for each setting of its inputs $\{z_{[1]}, d_{[1]}\}$. We use the single candidate design $\tilde{d}_2$ to evaluate the expectation (3.4.17) and the second term of the covariance (3.4.18) for each input setting, and we fix the first term of (3.4.18) to be constant across the input space. We fix its value by evaluating the expected risk surface $\mathrm{E}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\left[\tilde{d}_2\right]\right] + c_2\left(\tilde{d}_2\right)$ for 20 different candidate designs at each of 10 different settings of the inputs $\{z_{[1]}, d_{[1]}\}$. We compute the variance of the set of expectations for each of the 10 locations; the minimum variance that we obtain is $(0.0006)^2$, and the maximum is $(0.0013)^2$. On this basis, we judge that approximating this variance component as constant across the input space is reasonable, and we fix all variances $\mathrm{Var}\left[\mathrm{E}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\left[\tilde{d}_2\right]\right] + c_2\left(\tilde{d}_2\right)\right] = (0.0008)^2$, the mean of these variances. We set all covariances between expectation values at different inputs to zero.

3.4.10 Stopping

Suppose that at wave i, we have run the algorithm back to stage k ($1 \leq k < n$), having already carried out the first $(k-1)$ experiments at design settings $d_{[k-1]}$, observing $\{z_{[k-1]}, w_{[k-1]}\}$; based on the information available in our emulator $\bar{r}_k^{(i)}$, we must either choose make an immediate decision a_{k-1}^* based on our current beliefs about q, or choose to carry out the k^{th} experiment at a particular setting of the design parameters d_k.

If we knew all risk functions exactly, this choice would be simple: we would just compare ρ_{k-1}^t with ρ_k^*, and if $\rho_k^* < \rho_{k-1}^t$, we would carry out the k^{th} experiment at d_k^*; otherwise, we would make an immediate decision. However, since we are uncertain about ρ_k^*, we must make a choice which accounts for this uncertainty. Our strategy is as follows: first, we determine the design that we would choose if we were to carry out the k^{th} experiment; then, we assess whether, at this setting, we would commission the experiment or terminate; finally, we assess the maximum benefit that we might expect to gain from another wave of the procedure 2.

Choosing a design First, we choose a design for the k^{th} experiment; we denote the design that we choose by $\hat{d}_k$, and we select the $\hat{d}_k$ which minimises the sum of our expectation of the risk and the design cost

$$\hat{d}_k = \arg\min_{d_k} \left[\mathrm{E}_{R_k^{(i)}} \left[\bar{r}_k^{(i)} \left[d_k \right] \right] + c_k \left(d_k \right) \right] .$$

This optimisation problem is approximately solved either by using a suitable numerical optimisation procedure, or by interrogating the mean and cost surface using a large Latin hypercube of design inputs and selecting the setting which minimises the surface over this set.

Choosing a course of action Having fixed $\hat{d}_k$, we must choose whether we will actually perform the k^{th} experiment; we make this choice by comparing our expectation for the risk at $\hat{d}_k$ with the risk from an immediate decision. If

$$\mathrm{E}_{R_k^{(i)}} \left[\bar{r}_j^{(i)} \left[\hat{d}_k \right] \right] + c_k \left(\hat{d}_k \right) < \rho_k^t$$

then we carry out the k^{th} experiment at design setting $\hat{d}_k$; otherwise, we opt for an immediate decision, based on our current beliefs $p\left(q|z_{[k-1]}, w_{[k-1]}, d_{[k-1]}\right)$.

Assessing the value of further waves Having identified the course of action that we would take based on our current beliefs, we perform a simple assessment of the maximum value that we could gain from running another wave of the procedure, to learn more about the risks involved. If we knew the risk function $\bar{r}_k^{(i)}$ and the optimal design setting d_k^*, then the risk from an optimal course of action would be

$$\min\left[\rho_{k-1}^t, \bar{r}_k^{(i)}\left[d_k^*\right] + c_k\left(d_k^*\right)\right].$$

Now suppose that, once we have used our emulator to decide what to do, we discover that d_k^* is the true optimal design for this experiment; in this situation, our expectation for the loss incurred by deciding to experiment at $\hat{d}_k$ rather than at d_k^* is

$$\begin{aligned} v_k^{(i)} = &\min\left[\rho_{k-1}^t, \mathrm{E}_{R_k^{(i)}}\left[\bar{r}_k^{(i)}\left[\hat{d}_k\right]\right] + c_k\left(\hat{d}_k\right)\right] \\ &- \mathrm{E}\left[\min\left[\rho_{k-1}^t, \bar{r}_k^{(i)}\left[d_k^*\right] + c_k\left(d_k^*\right)\right]\right] \end{aligned} \tag{3.4.19}$$

where the expectation of the second term is approximated by sampling candidate designs $\tilde{d}_k$ as outlined in algorithm 3.

The quantity $v_k^{(i)}$ is the expected value of perfect information (EVPI) for the risk calculation. This is the amount that we expect to gain from exact knowledge of the risk function; as such it constitutes an upper bound on the amount that we should be willing to pay for further analysis of the risk function (as outlined in Section 2.2.2).

Suppose that the cost of another wave of analysis would be $c_{\text{wv}_{(i+1)}}$. If $c_{\text{wv}_{(i+1)}} \geq v_k^{(i)}$, then the cost of further analysis is greater than our expected gain from perfect knowledge of the risk; the additional knowledge that we might gain about the risk is not worth the amount that we would have to pay for it, and so we should just make a decision based on our current beliefs about the risks. Alternatively, if $c_{\text{wv}_{(i+1)}} < v_k^{(i)}$, then we must make a judgement about whether a further wave of analysis would provide enough information to be worth its cost. Suppose that after the $(i+1)^{\text{th}}$

wave, we believe that the EVPI will be about $\mathrm{E}\left[v_k^{(i+1)}\right]$; then if $c_{\mathrm{wv}_{(i+1)}} < v_k^{(i)} - \mathrm{E}\left[v_k^{(i+1)}\right]$, we believe that the next wave will be worth its cost.

In general, the cost $c_{\mathrm{wv}_{(i+1)}}$ of the computational resources required for another wave and our expectation $\mathrm{E}\left[v_k^{(i+1)}\right]$ for the EVPI after an additional wave may be difficult to assess; we do not attempt to do so for either of our two examples (Sections 3.4.12 and 4). For $\mathrm{E}\left[v_k^{(i+1)}\right]$, one strategy might be to make an approximate uncertainty specification for the risk $\bar{r}_k^{(i+1)}$ and then assess the EVPI as above.

3.4.11 Choosing inputs for risk evaluations

When fitting the emulators as described in Section 3.4.3, we must select a set of points $\{z_{[j]k}, w_{[j]k}, d_{[j]k}\}$ at which we evaluate the risk function to generate the data $R_{jk}^{(i)}$; naturally, we wish to do this in a way which provides the most useful information about the risk. At the first wave of the analysis ($i = 1$), we have no prior information about which design inputs are likely to be optimal, and so we simply wish to select the design input settings in a way which provides good coverage of the whole input space; at wave 1 therefore, we simply adopt the standard approach of using a space-filling Latin hypercube design in d_j (see, for example, Santner et al. [2002]), and then generating corresponding w_j and z_j by sampling from the distributions $p\left(w_j | w_{[j-1]}, d_{[j]}\right)$ and $p\left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]}\right)$.

At later waves, however, we have already fitted a sequence of emulators to the risk functions, and so we have access to information which could be used to inform the selection of the $\{z_{[j]k}, w_{[j]k}, d_{[j]k}\}$ at later stages. Specifically, we want to avoid evaluating the risk at points that we have already established are unlikely to be reached; that is, points d_j where it is unlikely that the risk ρ_j^* from future sampling will be preferable to the risk ρ_{j-1}^t from stopping and making a decision now. We therefore adopt the following design procedure for the later waves of analysis:

Wave $i = 2, 3, \ldots$: At later waves, design inputs d_j must satisfy both of the following criteria to be selected for the risk emulator:

- They are candidate designs $\tilde{d}_j$ (sampled as in Section 3.4.8) for the current setting of $\{z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\}$.

- They satisfy the following criterion (for $j > 1$)

$$P\left(\bar{r}_j^{(i-1)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}\right] + c_j\left(d_j\right) \leq \rho_{j-1}^t\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\right]\right) \geq 0.05 \ .$$

That is, there is a greater than 5% chance that, under this setting of d_j, we would choose to perform the experiment at stage j (under our beliefs about $r_j^{(i-1)}$ at wave $(i-1)$). While we do not have a full probabilistic description of $r_j^{((i-1))}$, the three-sigma rule of Pukelsheim [1994] allows us to derive an approximately equivalent second-order rule

$$\frac{\mathrm{E}\left[\bar{r}_j^{(i-1)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}\right]\right] + c_j\left(d_j\right) - \rho_{j-1}^t\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\right]}{\left[\mathrm{Var}\left[\bar{r}_j^{(i-1)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}\right]\right]\right]^{1/2}} \leq 3 \ . \tag{3.4.20}$$

After having generated design inputs which are likely to be informative, we complete the input set needed to run the risk by sampling w_j and z_j from $p\left(w_j | w_{[j-1]}, d_{[j]}\right)$ and $p\left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]}\right)$ respectively.

If, after exploring the space, we can find no design inputs which satisfy the above criteria, then (provided we are happy with our previously fitted emulators) this suggests that we will never choose to continue experimenting as far as stage j, under any settings of the inputs at the previous stages. This is an indication that we may want to re-think the specification of our original decision problem. If we no longer believe that we will ever carry out the experiments beyond stage j, then we should remove these from consideration, and solve only the resulting, simpler backward induction problem. Alternatively, if the option is open to us, we may want to replace the experiments that we were originally considering at stages j and above with cheaper and/or more informative alternatives, which we may actually deem worth performing.

3.4.12 Illustrative example, wave 1: remaining steps

Finally in this section, we complete our analysis of the simple, linear model example first introduced in Section 3.4.2, and discussed in Sections 3.4.5 3.4.7 and 3.4.9. In Section 3.4.9, our final action was to determine the procedure that will be used to

characterise uncertainty about $s_2^{(1)}$; in this Section, we begin our analysis of the risk function at the first stage ($j = 1$) of the design problem, for the first wave ($i = 1$) of the analysis.

Stage $j = 1$ First, we select the basis and covariance functions that we will use at this stage. Again, we relate our basis function specification to the risk itself; we choose a two-element basis, with $h_{11}^{(1)} = 1$, and

$$h_{12}^{(1)}\left(z_{[1]}, d_{[1]}\right) = \min\left[\rho_1^t\left[\tilde{z}_{[1]}, d_{[1]}\right], \mathrm{E}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\left[\tilde{z}_{[1]}, \bar{d}_{[2]}\right]\right] + c_2\left(\bar{d}_2\right)\right]$$

where $\tilde{z}_{[1]} = \mathrm{E}\left[z_1|d_1\right]$, and $\bar{d}_{[2]} = \{d_1, \bar{d}_2\}$, where $\bar{d}_2$ is a fixed, user-specified setting of the design parameter at stage 2 which is close to optimal for a wide range of input settings $\{z_{[1]}, d_{[1]}\}$. We fix $\bar{d}_2 = 0.68$ on the basis of output from the candidate design sampler.

For the covariance function, we opt for the same, separable, squared exponential form

$$\mathrm{Cov}\left[r_1^{(1)}\left[z_{[1]}, d_{[1]}\right], r_1^{(1)}\left[z'_{[1]}, d'_{[1]}\right]\right] = v_1^{(1)} c\left(d_1, d'_1|\lambda_d\right) c\left(z_1, z'_1|\lambda_z\right)$$

where $v_1^{(1)}$ is the marginal variance of the residual, and $c\left(.,.|\lambda\right)$ is a squared exponential correlation function (Appendix B.1).

Having specified these functions, we fit the emulator. When generating data for the update, we cannot evaluate $R_1^{(1)}$ exactly because of our uncertainty about $s_2^{(1)}$. As outlined in Section 3.4.4, we instead assess $\mathrm{E}\left[R_{1k}^{(1)}\right]$ and $\mathrm{Cov}\left[R_{1k}^{(1)}, R_{1l}^{(1)}\right]$ for each of the risk evaluations by evaluating $\rho_1^t\left[z_{[1]k}, d_{[1]k}\right]$, $\mathrm{E}\left[s_2^{(1)}\left[z_{[1]k}, d_{[1]k}\right]\right]$ and $\mathrm{Cov}\left[s_2^{(1)}\left[z_{[1]k}, d_{[1]k}\right], s_2^{(1)}\left[z_{[1]l}, d_{[1]l}\right]\right]$ for all input settings, assuming a multivariate Gaussian distribution for $s_2^{(1)}$ and assessing the moments of the expression (3.4.16) by sampling.

We fit the risk model $r_1^{(1)}$ as outlined in Section 3.4.4. First, we perform a regression using a set of 200 risk evaluations; using the output from this regression, we fix $\mathrm{E}\left[\alpha_{11}^{(1)}\right] = 0.0832$ and $\mathrm{E}\left[\alpha_{12}^{(1)}\right] = 0.611$, with $\mathrm{Var}\left[\alpha_{11}^{(1)}\right] = (3.11 \times 10^{-6})^2$ and $\mathrm{Var}\left[\alpha_{12}^{(1)}\right] = (1.60 \times 10^{-5})^2$. Using the residuals from the regression, we fix the marginal variance of the residual component to $v_1^{(1)} = (0.0178)^2$. We carry out the cross validation in the same way as described in Section 3.4.5, fixing the correlation

parameters $\{\lambda_d, \lambda_z\}$ of the covariance function to the setting from a Latin hypercube of such points which minimises the sum of predictive log-Gaussian likelihoods.

Lastly at this stage, we integrate our emulator as detailed in Section 3.4.6. As at the second wave (Section 3.4.7), because we were careful to eliminate dependence of the basis functions on z_1, integrating these is simple; we have that $\bar{h}_{1p}^{(1)} = h_{1p}^{(1)}$ for $p = 1, 2$. Also, since we have chosen an equivalent form for the covariance function, we integrate with respect to $p\left(z_1|d_1\right)$ in the same way as we integrated with respect to $p\left(z_2|z_{[1]}, d_{[2]}\right)$ in Section 3.4.7. The moments of the emulator $\bar{r}_1^{(1)}$ for the expected risk are plotted in Figure 3.6.

Stopping Now that we have completed the approximate backward induction procedure 2 for this problem, we assess the results and determine what we should do next, as outlined in Section 3.4.10. First, we identify what we would do if we were to continue experimenting; we interrogate the expected risk surface $\mathrm{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\left[d_1\right]\right] + c_1\left(d_1\right)$ at a Latin hypercube of 2000 design settings, and fix $\hat{d}_1 = -0.018$ to the setting which minimises the risk over these points. At this point, our expected risk is $\mathrm{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\left[\hat{d}_1\right]\right] + c_1\left(\hat{d}_1\right) = 0.2520$, with $\mathrm{Var}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\left[\hat{d}_1\right]\right] = (0.0006)^2$. We compare this with the risk $\rho_0^t = 0.635$ from an immediate decision based on our prior beliefs; since we are extremely confident that $\mathrm{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\left[\hat{d}_1\right]\right] + c_1\left(\hat{d}_1\right) < \rho_0^t$, we conclude that based on this analysis, we should perform the first experiment at $\hat{d}_1$.

We now compute the expected value of perfect information $v_1^{(1)}$ for the risk calculation, to help us assess whether another wave of our procedure would be appropriate. The EVPI is computed as in equation (3.4.19), where the expectation of the second component is approximated by sampling candidate designs as in Algorithm 3. Based on a sample of 200 candidate designs, we find that $v_1^{(1)} = 0.0011$. Given that this only a small fraction (around 0.4%) of the risk from the experiment that we would carry out, the potential gains from further analysis are relatively small, so unless our computer resources were cheap, it is unlikely that we would carry out another wave of analysis for this problem. We do not attempt to assess the cost of analysis here, and we opt to carry out a further wave of analysis to illustrate the procedure.

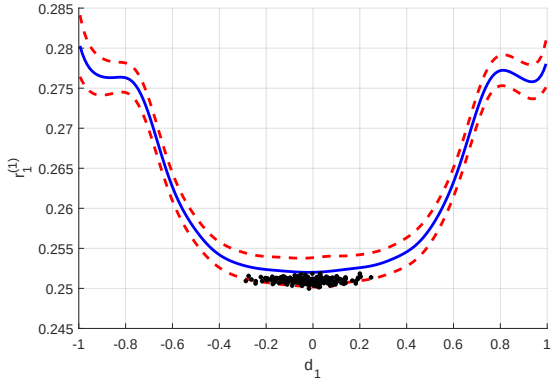
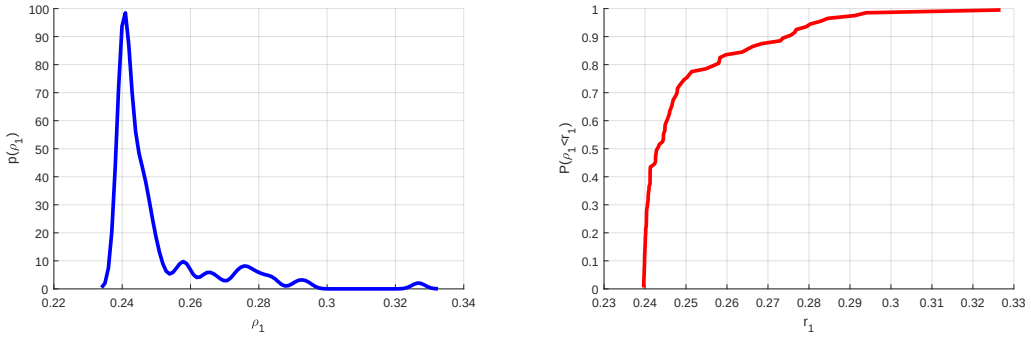


Figure 3.6: Plot of the beliefs about $\bar{r}_1^{(1)}[.]$ after wave 1 of the algorithm; $\mathrm{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\right]$ is shown in solid blue, and error bars $\mathrm{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\right] \pm 3 \times \mathrm{Var}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\right]$ are shown in dashed red. 200 candidate designs (generated according to algorithm 3) are shown in black.

As an additional diagnostic tool, we plot the risk profile of the procedure conducted according to these fitted emulators; designing the first experiment at $\hat{d}_1$, we generate z_1 from $p(z_1|d_1)$ and then use our fitted emulators to choose between performing the second experiment at $\tilde{d}_2$ (generated using the procedure 3) and making an immediate decision. We compute the actual terminal risk from each of the experimental procedures that we generate, and we plot the density of these risks. Figure 3.7(a) shows the density of a sample of 100 risks sampled in this way (generated using the 'ksdensity' function in Matlab), and Figure 3.7(b) shows the corresponding cumulative density.

3.4.13 Illustrative example, wave 2

Stage $j = 2$ Our first task at this wave is to fit the emulator $r_2^{(2)}$ inside the candidate design spaces from the first wave. To reduce the computational burden presented by the second wave, we characterise the candidate design spaces $\mathcal{D}_1^{(1)}$ and $\mathcal{D}_2^{(1)}$ by generating a single set of 200 pairs of candidate designs $\{\tilde{d}_1, \tilde{d}_2\}$ using the procedure 3, and restricting the designs that we generate for the fit at this wave to lie between the minimum and maximum values in the sample; this way, we restrict $d_1 \in [-0.31, 0.23]$ and $d_2 \in [-0.76, 0.76]$ at this wave.



(a) (b)

Figure 3.7: Risk profile for the experimental design procedure that begins with an experiment performed at $\hat{d}_1$: Figure 3.7(a) shows the approximate density of the resulting risks, and Figure 3.7(b) shows the approximate cumulative distribution function. Both plots were generated using 100 sampled trajectories, and the density was estimated using the Matlab function 'ksdensity'.

We judge that the basis and covariance functions that we used at the previous wave to have performed well, and so we use the same choices at this wave (for details, see Section 3.4.5). As in Sections 3.4.4 and 3.4.5, we fix the prior specification using an initial linear regression. We generate 200 evaluations of ρ_2^t for this regression, and based on the parameter estimates, we fix $\mathrm{E}\left[\alpha_{21}^{(2)}\right] = -2.29 \times 10^{-5}$ and $\mathrm{E}\left[\alpha_{22}^{(2)}\right] = 1.08$, with $\mathrm{Var}\left[\alpha_{21}^{(2)}\right] = (2.02 \times 10^{-7})^2$ and $\mathrm{Var}\left[\alpha_{22}^{(2)}\right] = (4.34 \times 10^{-6})$. Based on the residuals from this regression, we fix the marginal variance of the residual process to be $v_2^{(2)} = (0.0125)^2$. The usual leave-one-out cross-validation procedure is run, testing each point in a Latin hypercube of 2000 correlation parameter settings $\{\lambda_d, \lambda_z\}$ and keeping the one which performs best. 200 risk evaluations are used for the cross-validation and joint regression-residual update, and a further 100 are used for model checking after the fit.

Since we have used the same basis and covariance functions at this wave, the moments of $\bar{r}_2^{(2)}$ are also computed in the same way as at wave 1; for details of this calculation, see Section 3.4.7.

Stage $j = 1$ Moving back to the first stage of the calculation, the first thing that we must do is set up our procedure for assessing the moments (3.4.17) and (3.4.18) of the risk $s_2^{(2)}$ (see Section 3.4.8). Again, this is done in the same way as for the first wave (Section 3.4.9). We fix the size of our trial design set for the candidate design sampling procedure 3 to be $M_2^{(2)} = 200$, on the basis that this size worked well over a larger design space at the first wave. We then generate 20 candidate designs at each of 10 input settings, and fix the first component of the covariance (3.4.18) to $\text{Var}\left[\text{E}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\left[\tilde{d}_2\right]\right] + c_2\left(\tilde{d}_2\right)\right] = (0.0007)^2$, the average of the variance in the expectation across these input settings. All covariances between expectations at different input settings are set to zero.

Again, we use the same basis and covariance function specification for this stage at this wave as we did at wave 1. All risk moments for the fit are generated by sampling (3.4.16), using a multivariate Gaussian to draw samples of $s_2^{(1)}$ (see Section 3.4.12 for more detail), and we use the same basis and covariance functions as at the first wave. We generate 200 risk evaluations for the initial regression, fixing $\text{E}\left[\alpha_{11}^{(2)}\right] = 0.265$ and $\text{E}\left[\alpha_{12}^{(2)}\right] = -0.349$, with $\text{Var}\left[\alpha_{11}^{(2)}\right] = (1.09 \times 10^{-4})^2$ and $\text{Var}\left[\alpha_{12}^{(2)}\right] = (5.74 \times 10^{-4})^{-2}$; the residuals from the regression are used to fix the marginal variance of the residual to $v_1^{(2)} = (0.0156)^2$. 2000 possible correlation parameter settings $\{\lambda_d, \lambda_z\}$ (generated according to a Latin hypercube) are compared using the usual cross-validation procedure, and these parameters are fixed to the best setting for the joint update. After performing the joint update, a further 100 risk evaluations are used for model checking.

Since the basis and covariance functions used at this wave are the same as those used at wave 1, the moments of the expected risk $\bar{r}_1^{(2)}$ are also computed in the same way as before; see Sections 3.4.7 and 3.4.12 for further details.

Stopping Having run the second wave, we re-run the steps outlined in Section 3.4.10 to determine what we should now do. First, we interrogate the mean surface $\text{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}[d_1]\right] + c_1(d_1)$ at a Latin hypercube of 2000 design settings, and fix $\hat{d}_1 = 0.103$ to the setting which minimises the risk over these points. At this point, we have $\text{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\left[\hat{d}_1\right]\right] + c_1\left(\hat{d}_1\right) = 0.252$ and $\text{Var}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\left[\hat{d}_1\right]\right] = (0.0002)^2$.

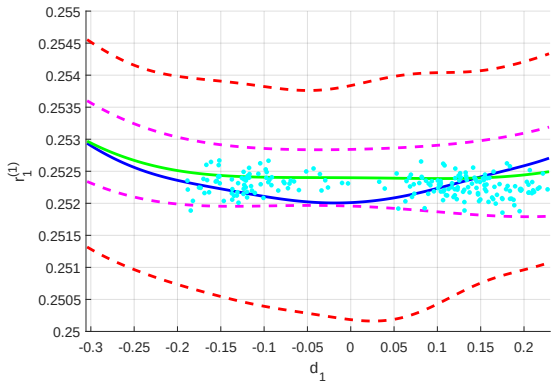
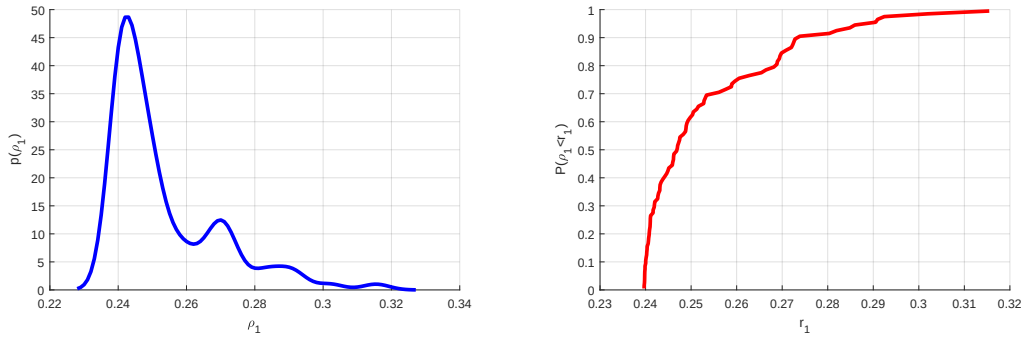


Figure 3.8: Plot of the emulators for $\bar{r}_1^{(i)}[.]$ resulting from both waves $i = 1, 2$ of the analysis; the solid blue line shows $\mathrm{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\right]$, and the dashed red lines show the error bars $\mathrm{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\right] \pm 3\mathrm{Var}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\right]^{1/2}$; the corresponding means and error bars from wave 2 are shown in green and magenta respectively. Candidate designs $\tilde{d}_1$ generated from the emulator at the second wave are shown as cyan markers.

We again compute the EVPI for the risk calculation, to help us assess the value of further computation. By evaluating the expectation of the second term in (3.4.19) by generating 200 candidate designs $\tilde{d}_1$ using algorithm 3, we assess that $v_1^{(2)} = 0.0001$ after this wave. There has been a small reduction in the EVPI value between waves 1 and 2, owing to a reduction in our uncertainty about the risk within the candidate design space (as demonstrated in Figure 3.8). As before, whether we choose to carry out further analysis based on this value depends on the relative cost of the computer resources required; for the purposes of illustrating the algorithm, however, we stop here.

In Figure 3.9, we plot the risk profile for the second wave of the analysis, again using 100 risks from experimental trajectories sampled as outlined in Section 3.4.12. We see by comparing Figures 3.7 and 3.9 that the distribution of risks that we might expect from designing experiments according to our fitted emulators has changed little between waves of analysis.



(a) (b)

Figure 3.9: Risk profile plot for the second wave of the analysis, for a procedure which begins with an experiment at $\hat{d}_1$: Figure 3.9(a) shows the approximate density, and Figure 3.9(b) shows the approximate cumulative density. Both plots were generated using the same sample of 100 trajectories.

Chapter 4

Example: atmospheric dispersion problem

In this section, we apply the approximate backward induction procedure to a more complex example, in which we must estimate the locations and emission rates of a particular gas species using concentration measurements obtained from an airbourne sensor. In this problem, the forward model has a more complex dependence on its input parameters, and the design and data spaces are much higher dimensional. In Section 4.1.1, we set up the forward model for this problem by combining the Gaussian plume model introduced in Section 1.1.3 with a parametric form for the flight path of the plane. Then, in Section 4.1.2, we present the decision problem that we will solve, and outline the calculations that we will need to perform in order to adjust our prior beliefs about the model parameters. Then in Section 4.2, we run algorithm 2 for a three-stage version of this problem, discussing the results of our analysis. Finally, in Section 4.3, we consider some of the practical difficulties encountered, and outline future work which we feel would be helpful in overcoming these issues. Some of the code used to implement this example is supplied in Appendix E.

4.1 Atmospheric dispersion problem

We now consider the application of the sequential design algorithm from Section 3.4 to a more complex atmospheric modelling problem. The forward model in this problem links gas emission rates at a grid of points on the ground to concentration measurements made on the flight path of an aircraft flown above the region using the Gaussian plume outlined in Section 1.1.3. The observations made by the aircraft are then to be used to make decisions about the source emission rates on the ground. Since the flights are costly, we want to have the ability to re-evaluate our state of information after each data set has been obtained, and to stop sampling if it is unlikely that any further flights will provide enough of a reduction in the risk to offset their cost.

In Section 4.1.1, we set up our model for the observed concentration data. Then, in Section 4.1.2, we introduce a parametrisation for the flight paths in terms of a small number of key parameters and combine this with our model for the individual data points and introduce the decision problem that we will solve using our posterior beliefs. Then, in Section 4.2, we run the approximating algorithm for a 3-stage problem, and interpret the results.

4.1.1 Forward model

Our model for the observed concentration data is based on the Gaussian plume outlined in Section 1.1.3; the contribution made by a set of sources $g = \{g_1, \ldots, g_{n_\psi}\}$ with corresponding emission rates $\psi = \{\psi_1, \ldots, \psi_{n_\psi}\}$ to the concentration at a location x under wind conditions w is assumed to be

$$u\left(x, g, \psi, w\right) = \sum_{j=1}^{n_\psi} a\left(\omega(x, g_j, w), g_j, w\right) \psi_j$$

where the wind-projected distances $\omega(.)$ and the Gaussian plume coupling coefficients $a\left(.\right)$ are computed as in Section 1.1.3. The source locations $\{g_j\}$ are fixed at the centres of a grid at ground level within a rectangular domain, and we seek to estimate the emission rates $\{\psi_j\}$ associated with each location.

Each flight path is assumed to be composed of a series of n_{fl} straight lines, parametrised

in terms of a set of five design variables d, where:

- $\{d_x, d_y\}$ (metres) determine the centre of the flight path in the horizontal plane;
- d_h (metres) determines the altitude of the aircraft for the whole data-collection period of the flight;
- d_w (metres) determines the length of each flight line;
- d_d (metres) determines the distance between the flight-lines.

We also specify the cruising speed of the observing aircraft, and the frequency with which the measuring equipment mounted on the plane will record data; based on this information, a deterministic mapping $x_p = \xi_p(d)$ then determines the relationship between this 5-dimensional parametrisation of each flight and the 3-dimensional spatial locations of the observations that will actually be collected.
Combining this description of the flight path with the representation of the concentration observations, and identifying the model parameters $q = \psi$ with the emission rates, we obtain our model for the data $\{z_k\}$ observed at d

$$z_k = \sum_{l=1}^{n_\psi} A_{kl}(w, d)\, q_l + \epsilon_k \tag{4.1.1}$$

where the $\{\epsilon_k\}$ are measurement error terms, which are assumed to be independent of the parameters q and all other inputs, and the $A_{kl}(w, d) = a(\omega(\xi_k(d), g_l, w), g_l, w)$ are the components of a coupling matrix which links the emission rates to the observed data. We make an uncertainty specification for $\{q, \epsilon\}$ in Section 4.1.2.

4.1.2 Decision problem and belief adjustment

Decision problem We combine this model for the concentration with a specification for the decision problem that we will solve using our posterior beliefs; we specify that we make a decision about each individual source, and that our loss function decomposes into a sum of losses corresponding to each individual decision

$$L(q, a) = \sum_{k=1}^{n_\psi} l_k(q_k, a_k)$$

where we specify that each of the individual components of the loss function is a weighted quadratic loss (Section 3.1.1)

$$l_k\left(q_k, a_k\right) = \gamma\left(q_k\right)\left(q_k - a_k\right)^2 .$$

The optimal decision a^* is found by choosing the decision which minimises each individual component of the loss; recall from Section 3.1.1 that each optimal decision is

$$a_k^* = \frac{1}{\mathrm{E}\left[\gamma\left(q_k\right)\right]}\mathrm{E}\left[\gamma\left(q_k\right) q_k\right] .$$

After observing $z_{[j]}$ (at $\{w_{[j]}, d_{[j]}\}$) and computing our posterior beliefs about q_k, the terminal risk from an optimal decision is made up of corresponding individual risk components

$$\rho_j^t\left[z_{[j]}, w_{[j]}, d_{[j]}\right] = \sum_{k=1}^{n_\psi} \rho_{jk}^t\left[z_{[j]}, w_{[j]}, d_{[j]}\right]$$

where

$$\rho_{jk}^t\left[z_{[j]}, w_{[j]}, d_{[j]}\right] = \mathrm{E}\left[\gamma\left(q_k\right) q_k^2\right] - \frac{1}{\mathrm{E}\left[\gamma\left(q_k\right)\right]}\mathrm{E}\left[\gamma\left(q_k\right) q_k\right]^2 \tag{4.1.2}$$

where the expectations are computed using the posterior marginal distributions $p\left(q_k | z_{[j]}, w_{[j]}, d_{[j]}\right)$.

In this problem, the cost of the observations is determined by the cost of the flying the plane (e.g. fuel, rental, manpower). Instead of attempting to work out the exact length of the flight path (which we would almost certainly get wrong, since the actual path taken will depend on the atmospheric conditions on the day), we use a simple approximation to its length, comprising the length of the return trip to the centre of the flight path and the horizontal distance flown during the survey. In terms of the 5-dimensional parametrisation of the flight path introduced earlier, the cost of a flight is

$$c\left(d\right) = c_{\mathrm{st}} + c_{\mathrm{fl}}\left[2\left((d_x - x_x^{(0)})^2 + (d_y - x_y^{(0)})^2\right)^{1/2} + (n_{fl} - 1)d_d + n_{fl}d_w\right]$$

where c_{fl} is the cost per metre of the flight path, $x^{(0)}$ is the location of the airport at which the aircraft will take-off and land, and c_{st} is a constant set-up cost. The cost of a flight is assumed to be the same for all stages of the problem, i.e. $c_j\left(d_j\right) = c\left(d_j\right)$.

Inference A number of different fully probabilistic prior specifications could be used for this model. At the simplest level, we could assume a multivariate Gaussian prior distribution for the emission rates; if we were to do this, the linear forms of all of the relationships specified in Section 4.1.1 would mean that we could simply make a joint Gaussian specification for all components (conditional on any correlation parameters and marginal variances for the discrepancy and measurement error terms), marginalise over the discrepancy and the error distributions and compute our posterior beliefs about q, resulting in another multivariate Gaussian.

Hirst et al. [2013] raise a number of issues with this approach. First, the use of a multivariate Gaussian distribution does not constrain the source emission rates to be positive; an emission rate $q_k < 0$ would correspond to a gas sink (which absorbs gas from the atmosphere), and while this is physically possible, we do not believe that there are likely to be any sinks within a typical survey area. Additionally, we expect sources of gas to be localised. We do not expect low emissions from a wide range of ground locations; rather, we expect larger emission rates from a smaller number of constrained areas.

Procedures exist for imposing these characteristics on sources. Hirst et al. [2013] use a model with an unknown number of circular sources with fixed radii, with a prior uniform distribution for the emission rates which imposes that $q_k \geq 0$; however, since we do not know the number of sources in the domain, this must be estimated from the observed data using a reversible jump MCMC scheme (Green [1995]), alongside all of the other components of the model. The need to use such a computationally expensive MCMC scheme for inference would make the algorithm in Section 3.4 computationally infeasible. Indeed, even imposing the positivity requirement for the emission rates within a probabilistic framework would remove tractability and introduce the need for a large MCMC scheme for inference.

For the purposes of the design calculation, we opt instead to make a second-order specification for each of the model components in 4.1.1 and to carry out inference by performing a Bayes linear adjustment upon observation of the data; we impose the positivity requirement for emission rates by simply setting any emission rates for which $q_k < 0$ to zero before computing the corresponding risk. While this second-

order specification does not capture our beliefs about localisation and positivity, it gives a simple characterisation of our uncertainty, which we believe will be sufficient for assessment of suitable designs.

Using (4.1.1), our model for the components of the data $z_j = \{z_{j1}, \ldots, z_{jn_{z_j}}\}$ observed at experiment j at design setting d_j, under external conditions w_j (the wind field at stage j) is

$$z_{jl} = \sum_{p=1}^{n_\psi} A_{lp}\left(w_j, d_j\right) q_p + \epsilon_{jl} \ .$$

We make a second-order prior specification for the emission rates and the measurement error terms, consisting of components $\mathrm{E}\left[q_p\right]$, $\mathrm{Cov}\left[q_p, q_q\right]$ and $\mathrm{Var}\left[\epsilon_{jk}\right]$ (the ϵ_{jk} are assumed to be uncorrelated with each other); re-introducing the Einstein summation convention, our corresponding second-order prior specification for the data is then

$$\begin{aligned}
\mathrm{E}\left[z_{jl}\right] &= A_{lp}\left(w_j, d_j\right) \mathrm{E}\left[q_p\right] \\
\mathrm{Cov}\left[z_{jl}, z_{kr}\right] &= A_{lp}\left(w_j, d_j\right) \mathrm{Cov}\left[q_p, q_t\right] A_{rt}\left(w_k, d_k\right) \\
&\quad + \mathrm{Cov}\left[\epsilon_{jl}, \epsilon_{kr}\right] \ .
\end{aligned}$$

After observing the data $z_{[j]} = \{z_1, \ldots, z_j\}$ from the first j experiments, we adjust our beliefs about the emission rate parameters q as follows

$$\begin{aligned}
\mathrm{E}_{z_{[j]}}\left[q_p\right] &= \mathrm{E}\left[q_p\right] + \mathrm{Cov}\left[q_p, z_{tu}\right] \mathrm{Var}\left[z_{[j]} | w_{[j]}, d_{[j]}\right]^{-1}_{tuvw} \Big[z_{vw} - \mathrm{E}\left[z_{vw}\right] \Big] \\
\mathrm{Cov}_{z_{[j]}}\left[q_p, q_r\right] &= \mathrm{Cov}\left[q_p, q_r\right] - \mathrm{Cov}\left[q_p, z_{tu}\right] \mathrm{Var}\left[z_{[j]} | w_{[j]}, d_{[j]}\right]^{-1}_{tuvw} \mathrm{Cov}\left[z_{vw}, q_r\right]
\end{aligned}$$

where the covariances between the data and the model parameters are

$$\mathrm{Cov}\left[q_p, z_{jl}\right] = \mathrm{Cov}\left[q_p, q_t\right] A_{lt}\left(w_j, d_j\right) \ .$$

We use these adjusted moments to evaluate the components (4.1.2) of the risk function; where evaluating these components requires the specification of higher-order moments, we obtain these from a Gaussian distribution characterised by our adjusted expectation and variance.

For the remainder of this chapter, we fix our prior expectations and covariances for the emission rates as follows, corresponding to the mean and covariance specifications for a log-Gaussian distribution

$$\begin{aligned} \mathrm{E}\left[q_p\right] &= \exp\left(\mu_p\right) \\ \mathrm{E}\left[q_p, q_r\right] &= \exp\Big(\mu_p + \mu_r + \frac{1}{2}\big(\Sigma_{pp} + \Sigma_{rr}\big)\Big)\Big(\exp\big(\Sigma_{pr}\big) - 1\Big) \end{aligned}$$

where $\mu_p = -4$, and

$$\Sigma_{pr} = (2^2)\exp\Big(-\frac{1}{2(1000^2)}(g_p - g_r)^2\Big) \ .$$

Additionally, we fix $\mathrm{Var}\left[\epsilon_{jp}\right] = (1 \times 10^{-8})$ for all measurement error terms.

Figure 4.1 shows a set of expected concentrations generated under this model for fixed q, and the corresponding inference; Figures 4.1(a) to 4.1(c) show the expected concentrations generated at an altitude of 200 metres by the sources located at the magenta markers (both of which have a true emission rate of $q_p = 1$) under three different wind vectors (indicated by the black arrows), alongside flight paths which are used to collect data in each instance. Figures 4.1(d) and 4.1(e) show our adjusted expectations $\mathrm{E}_{z_{[j]}}\left[q|w_{[j]}, d_{[j]}\right]$ and variances $\mathrm{Var}_{z_{[j]}}\left[q|w_{[j]}, d_{[j]}\right]$ for the source emission rates given the data observed on the displayed flight paths.

4.2 Running the algorithm

Using the model and decision problem specifications outlined above, we run the algorithm outlined in Section 3.4 for a three-stage version of the problem; we may commission a maximum of 3 flights over the survey area, and we wish to determine, after the data from each flight has been analysed, whether we should pay for another one, or stop and make a decision using the data the we have already obtained.

Parameter specification: For this run of the algorithm, we fix the source domain to be the box $[0, 5000] \times [0, 5000]$, and we impose an even 10×10 grid of locations $\{g_k\}$. We fix the limits of the design space as follows:

$$\begin{aligned} &d_x \in [0, 5000] \qquad d_h \in [100, 300] \qquad\qquad d_d \in [100, 500] \\ &d_y \in [0, 5000] \qquad d_w \in [500, 1000] \ . \end{aligned}$$

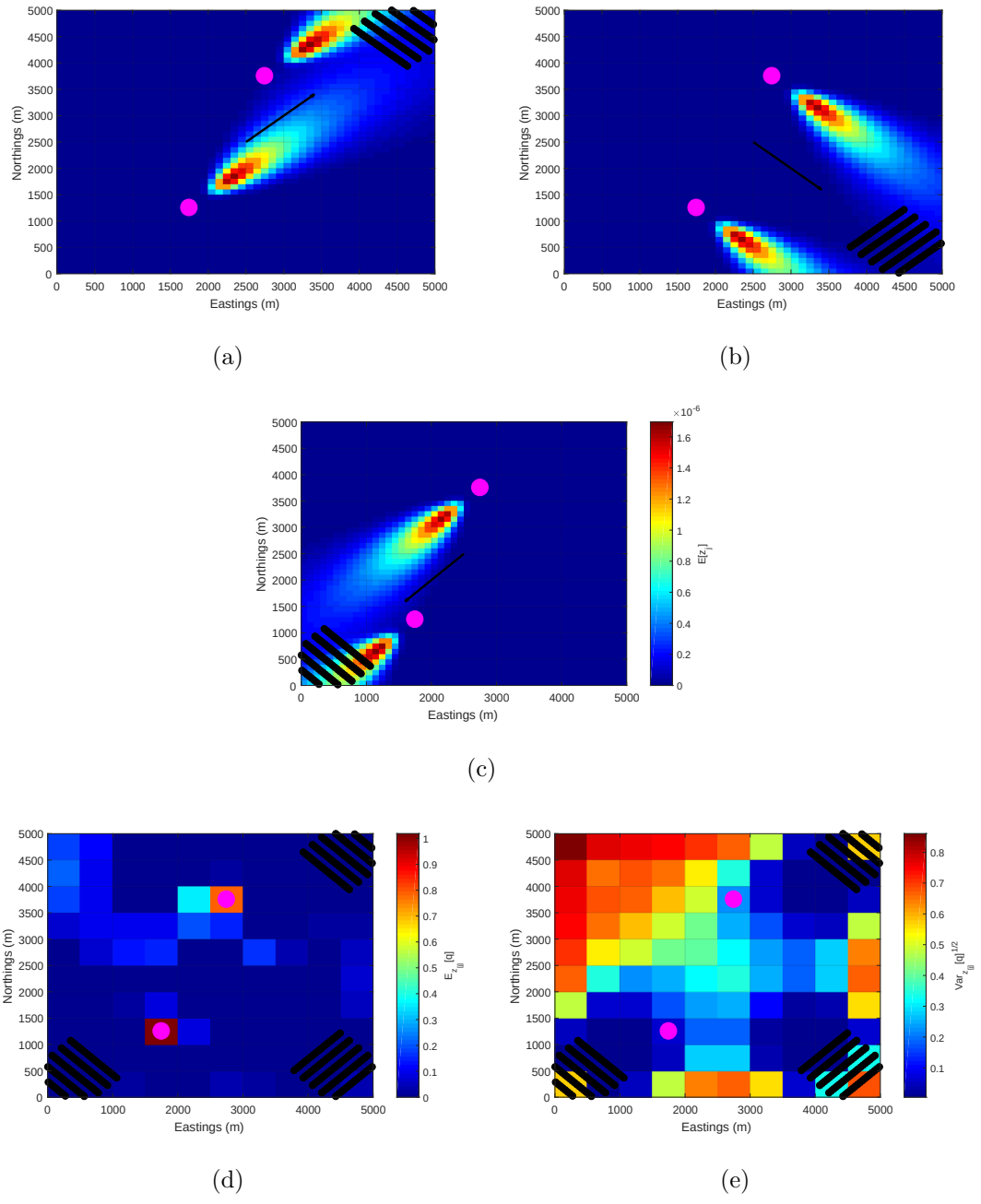


Figure 4.1: Expected concentrations $\mathrm{E}\,[z]$ at an altitude of 200 metres for the sources located at the magenta markers (colour scales in Figures 4.1(a) to 4.1(c)), under wind conditions indicated by the black arrows, and the corresponding adjusted moments $\mathrm{E}_{z_{[j]}}\,[q]$ (Figure 4.1(d)) and $\mathrm{Var}_{z_{[j]}}\,[q]^{\,1/2}$ (Figure 4.1(e)) for the source emission rates given data $z_{[j]}$ observed on the flight paths shown.

That is, we allow the flight path to be centred at any point vertically above the source domain, between 100 and 300 metres above ground, and we allow the length of an individual flight line to vary between 500 and 1000 metres, with between 100 and 500 metres between flight lines. We specify that an aircraft will carry out 5 flight lines, and that it will make 20 observations along the length of each flight line. We specify that our beliefs about the wind field are different at each stage of the problem (all units are m/s):

- at stage $j = 1$, $(w_x, w_y) \in [-3, -2] \times [2, 3]$;
- at stage $j = 2$, $(w_x, w_y) \in [2, 3] \times [-3, -2]$;
- at stage $j = 3$, $(w_x, w_y) \in [-3, -2] \times [-3, -2]$.

We do not impose a dependence of the distribution of winds on the flight-path design, and we specify an independent uniform distribution for each individual component of the wind at each stage, so

$$p\left(w_j|d_j\right) = p\left(w_j\right) = p\left(w_{jx}\right) p\left(w_{jy}\right) = \mathcal{U}\left(\left[l_{jx}, u_{jx}\right]\right) \mathcal{U}\left(\left[l_{jy}, u_{jy}\right]\right)$$

where the limits l_{jx}, u_{jx} etc are specified above.
Finally, we specify the components of the decision problem. For the flight costs, we specify $c_{\text{fl}} = 1 \times 10^{-4}$ and $c_{\text{st}} = 2$, and for the weighting for the quadratic loss function, we choose

$$\gamma\left(q\right) = 2 \ .$$

Having specified all of these components, we are ready to run the algorithm. In Section 4.2.1, we outline the basis and covariance function specifications that we use for our risk emulators. Then, in Section 4.2.2, we provide details relating to the first wave of analysis, and in Section 4.2.3, we provide details relating to a second wave.

4.2.1 Basis and covariance functions

The basis and covariance function specifications that we use are common across both waves; we provide details relating to both in this section.

Basis functions For all risk functions, we use two basis functions; the first is an intercept term $h_{j1}^{(i)} = 1$, and the second is related to our current beliefs about the risk. At the final stage ($j = n = 3$), we choose

$$h_{n2}^{(i)}\left(z_{[n]}, w_{[n]}, d_{[n]}\right) = \rho_n^t\left[\tilde{z}_{[n]}, \tilde{w}_{[n]}, d_{[n]}\right]$$

so that the basis function is the terminal risk with the inputs $z_{[n]}$ and $w_{[n]}$ replaced with $\tilde{z}_{[n]} = \{z_1, z_2, \mathrm{E}_{z_{[2]}}\left[z_3|\tilde{w}_{[3]}, d_{[3]}\right]\}$ and $\tilde{w}_{[n]} = \{w_1, w_2, \mathrm{E}\left[w_3\right]\}$. At stages $j = 1, 2$, we set

$$h_{j2}^{(i)}\left(z_{[j]}, w_{[j]}, d_{[j]}\right) = \min\left[\rho_j^t\left[\tilde{z}_{[j]}, \tilde{w}_{[j]}, d_{[j]}\right], \mathrm{E}_{R_{j+1}^{(i)}}\left[\bar{r}_j^{(i)}\left[\tilde{z}_{[j]}, \tilde{w}_{[j]}, \bar{d}_{[j+1]}\right]\right] + c_{j+1}\left(\bar{d}_{j+1}\right)\right]$$

where $\tilde{z}_{[j]}$ and $\tilde{w}_{[j]}$ are defined in the same way as above, replacing z_j and w_j with their expectations, and $\bar{d}_{[j+1]} = \{d_1, \ldots, \bar{d}_{j+1}\}$, where $\bar{d}_{j+1}$ is a design parameter setting which we fix to a value that we believe will be close to optimal for a wide variety of different input settings $\{z_{[j]}, w_{[j]}, d_{[j]}\}$.

These forms for the basis functions have been carefully selected so that they have no dependence on the inputs $\{z_j, w_j\}$ over which we must integrate. When computing the moments of $\bar{r}_j^{(i)}$ (as outlined in Section 3.4.6), we simply have that $\bar{h}_{j1}^{(i)} = h_{j1}^{(i)}$ and $\bar{h}_{j2}^{(i)} = h_{j2}^{(i)}$.

For this example, the use of the above risk function greatly improves the ability of the fitted emulator to capture the non-linear and interacting behaviour of the risk, compared with the use of a traditional polynomial basis. The alteration of the input terms to remove dependencies on any inputs that we must integrate over allows us to build a regression surface which captures a substantial amount of the global behaviour of the risk, while still retaining the ability to compute integrals of the adjusted emulator moments in closed-form.

Covariance functions Because of our choice of a constant weight function for the loss, none of the risk functions depend on the data $z_{[j]}$; for our covariance functions,

then, we choose the following separable form for all waves i and stages j

$$\mathrm{Cov}\left[u_j^{(i)}\left(z_{[j]}, w_{[j]}, d_{[j]}\right), u_j^{(i)}\left(z'_{[j]}, w'_{[j]}, d'_{[j]}\right)\right] = v_j^{(i)} \prod_{k=1}^{j}\left[\left[\prod_{p=1}^{5} c\left(d_{jp}, d'_{jp}|\lambda_{d_p}^{(i)}\right)\right]\left[\prod_{p=1}^{2} c\left(w_{jp}, w'_{jp}|\lambda_{w_p}^{(i)}\right)\right]\right]$$

where $c\left(.,.|\lambda\right)$ is a scalar squared exponential (SE) correlation function (see Appendix B.1) with correlation parameter λ, $v_j^{(i)}$ is the marginal variance of the process at the j^{th} stage and i^{th} wave, and $\lambda_{d_p}^{(i)}$ is the correlation parameter corresponding to the design parameter d_{jp} for all stages j at wave i. As discussed in Section 3.4.4, for each emulator fit, the marginal variance of the residual process is fixed to the variance of the residuals from the initial regression fit, and the correlation parameters are fixed using leave-one-out cross-validation.

After fitting each emulator $r_j^{(i)}$, we use this to compute beliefs about the expectation $\bar{r}_j^{(i)}$ of the risk; in order to compute the moments outlined in Section 3.4.6, we must integrate this covariance function with respect to $p\left(w_j\right)$. As outlined at the start of Section 4.2, the external parameter components are independently uniformly distributed at each stage, and so we have that

$$\mathrm{Cov}\left[\bar{u}_j^{(i)}\left(z_{[j-1]}, w_{[j-1]}, d_{[j]}\right), u_j^{(i)}\left(z'_{[j]}, w'_{[j]}, d'_{[j]}\right)\right] = v_j^{(i)} \prod_{k=1}^{j}\left[\left[\prod_{p=1}^{5} c\left(d_{jp}, d'_{jp}|\lambda_{d_p}^{(i)}\right)\right]\left[\prod_{p=1}^{2} \bar{c}\left(w'_{jp}|\lambda_{w_p}^{(i)}\right)\right]\right]$$

and that

$$\mathrm{Cov}\left[\bar{u}_j^{(i)}\left(z_{[j-1]}, w_{[j-1]}, d_{[j]}\right), \bar{u}_j^{(i)}\left(z'_{[j-1]}, w'_{[j-1]}, d'_{[j]}\right)\right] = v_j^{(i)} \prod_{k=1}^{j}\left[\left[\prod_{p=1}^{5} c\left(d_{jp}, d'_{jp}|\lambda_{d_p}^{(i)}\right)\right]\left[\prod_{p=1}^{2} \bar{\bar{c}}\left(\lambda_{w_p}^{(i)}\right)\right]\right].$$

The functions $\bar{c}\left(.|\lambda_{w_p}^{(i)}\right)$ and $\bar{\bar{c}}\left(\lambda_{w_p}^{(i)}\right)$ are computed by integrating the SE function as outlined in Appendix B.1.2.

4.2.2 First wave

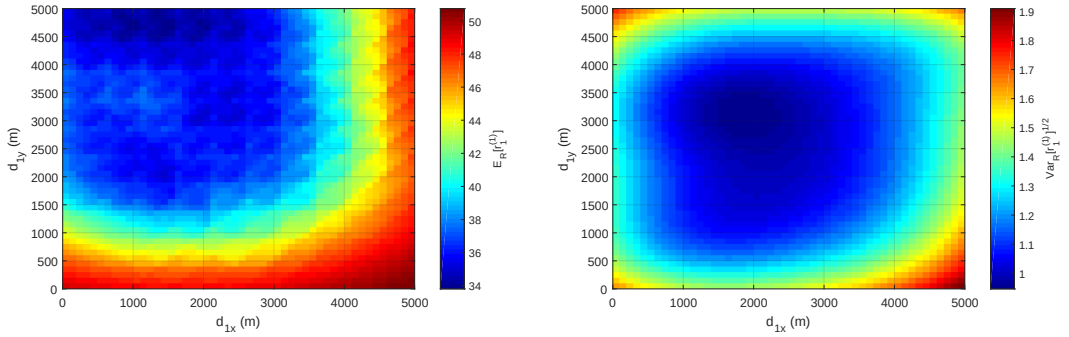
In this section, we run a first wave of the procedure outlined in Section 3.4, providing details of the emulator fits and discussing the results. At all stages $j = 1, 2, 3,$

the fitted risk emulator is checked against an additional set of risk evaluations to ensure that fewer than 5% of the new points lie outside the three-standard deviation predictive error bars generated by the emulator.

Stage $j = 3$ We follow the general emulator fitting procedure outlined in Section 3.4.4; this being the final stage, the data $R_3^{(1)}$ for the emulator update are generated directly from the terminal risk (equation (3.3.6)), which can be evaluated quickly as outlined in Section 4.1. We generate 200 risk evaluations for the initial regression, 500 for the joint adjustment of the regression surface and the residual component, and 100 for post-fit checking of our emulator. Performing the initial regression, we find that $\mathrm{E}\left[\alpha_{31}^{(1)}\right] = 0.535$ and $\mathrm{E}\left[\alpha_{32}^{(1)}\right]$, with $\mathrm{Var}\left[\alpha_{31}^{(1)}\right] = (2.5 \times 10^{-3})^2$ and $\mathrm{Var}\left[\alpha_{32}^{(1)}\right] = (2.8 \times 10^{-5})^2$. Using the residuals from this regression surface, we fix $\mathrm{Var}\left[u_3^{(1)}\right] = v_3^{(1)} = (2.09)^2$.

Next, we determine the correlation parameters for the covariance function using leave-one-out cross validation. We treat the regression surface as fixed at its mean level, and subtract these mean predictions from the risk evaluations; the cross-validation procedure is then run for these regression-subtracted values. We compare the quality of the prediction for each individual point using the fit to the remainder for a Latin hypercube of 2000 $\{\lambda_d^{(i)}, \lambda_w^{(i)}\}$ values, using the sum of predictive log-Gaussian likelihoods as our criterion for the comparison. We fix the correlation parameters to the best performing setting. Having fixed the correlation parameters, we are in a position to be able to compute adjusted moments for new input settings, as outlined in Section 2.2.2. The fitted emulator is checked against the 100 additional points generated earlier.

Two sets of predictions from the fitted emulator are shown in Figures 4.2 and 4.3; in both figures, the risk is predicted at a grid of (d_{3x}, d_{3y}) values for fixed settings of the flight design at stages 1 and 2. We use a different setting of (d_{1x}, d_{1y}) (flight path shown in black markers) and (d_{2x}, d_{2y}) (magenta markers) in each Figure. In Figure 4.2, the chosen designs are downwind of the majority of sources in both cases, so the observations that we make cause a substantial reduction in our uncertainty, resulting in lower risks; however, in Figure 4.3, the designs selected in each instance

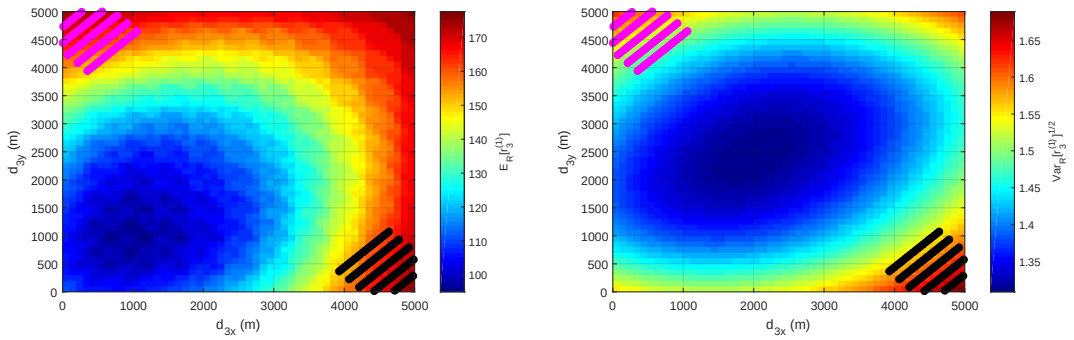


(a) (b)

Figure 4.2: Plots of the adjusted emulator moments for the emulator fitted at the final stage (wave $i = 1$). The black markers show the locations of the observations made at stage 1, and the magenta markers show the locations of the observations at stage 2; the colour scale in Figure 4.2(a) represents the expected risk $\mathrm{E}_{R_3^{(1)}}\left[r_3^{(1)}\right]$ for varying (d_{3x}, d_{3y}), and the colour scale in Figure 4.2(b) represents the standard deviation $\mathrm{Var}_{R_3^{(1)}}\left[r_3^{(1)}\right]^{1/2}$ for the same points. The remaining design parameters are fixed to $d_{jh} = 200$, $d_{jw} = 1000$ and $d_{jd} = 200$ for all stages.

are upwind of the majority of sources, resulting in a small reduction in uncertainty, and higher risks.

Stage $j = 2$ Moving back to consider the risk at stage $j = 2$, our first task is to specify how we will approximate the moments of the risk $s_3^{(1)}$ from an optimal design at the final stage. The candidate design space is characterised as outlined in Section 3.4.8 (Algorithm 3). Sampling multiple candidate designs for each setting of $\{z_{[2]}, w_{[2]}, d_{[2]}\}$ would greatly increase the computational complexity of the fitting procedure; we therefore opt to sample only a single candidate design $\tilde{d}_3$ for each input setting, to approximate the expectation (3.4.17) using the expectation of $\bar{r}_3^{(1)}$ at this point, and to approximate the second term in the covariance (3.4.18) using the covariance between $\bar{r}_3^{(1)}$ values at pairs of $\tilde{d}_3$ values. We approximate the second contribution to the covariance (3.4.18) by fixing it to be constant across the inputs space. To do this, we generate 20 candidate designs at each of 10 different input settings and compute the variance of $\mathrm{E}_{R_3^{(1)}}\left[\bar{r}_3^{(1)}\right] + c_3\left(\tilde{d}_3\right)$ at each setting of

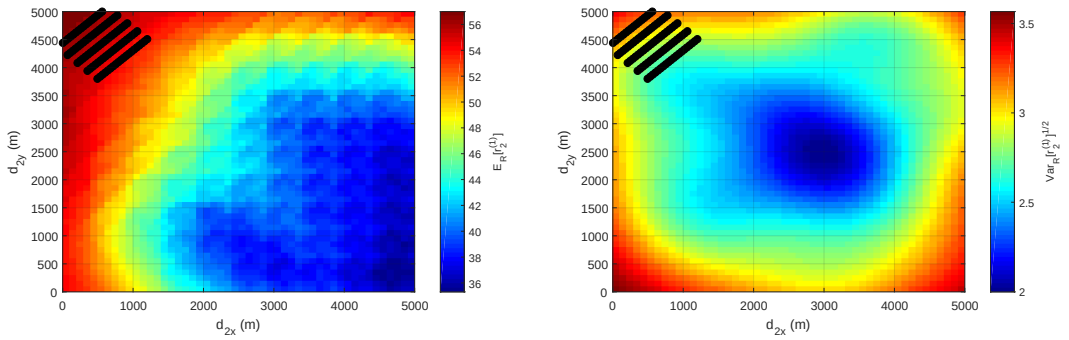


(a) (b)

Figure 4.3: Plots of adjusted moments for the emulator $r_3^{(1)}$. Figure 4.3(a) shows the expectation $\mathrm{E}_{R_3^{(1)}}\left[r_3^{(1)}\right]$ and Figure 4.3(b) shows the standard deviation $\mathrm{Var}_{R_3^{(1)}}\left[r_3^{(1)}\right]^{1/2}$; colour scales, marker colours and $\{d_{jh}, d_{jw}, d_{jd}\}$ settings correspond between figures. Risks are predicted at the same design input settings (d_{3x}, d_{3y}) as in Figure 4.2, but the designs for stages 1 and 2 are switched; both design settings give poor coverage of the survey area, and so the predicted risks are correspondingly higher.

$\{z_{[2]}, w_{[2]}, d_{[2]}\}$; we then fix $\mathrm{Var}\left[\mathrm{E}_{R_3^{(1)}}\left[\bar{r}_3^{(1)}\left[\tilde{d}_3\right]\right] + c_3\left(\tilde{d}_3\right)\right] = (1.99)^2$, the average variance across these 10 input locations, and fix all covariances between different input settings to zero.

Having fixed our characterisation of $s_3^{(1)}$ for all input settings, we proceed to fit the emulator. We generate a set of 100 risk evaluations for the initial regression, a set of 300 evaluations for the joint regression surface-residual update, and a set of 100 for checking the fitted emulator. Performing the initial regression, we fix $\mathrm{E}\left[\alpha_{21}^{(1)}\right] = 8.39$ and $\mathrm{E}\left[\alpha_{22}^{(1)}\right] = 0.62$, with $\mathrm{Var}\left[\alpha_{21}^{(1)}\right] = (4.47 \times 10^{-7})^2$ and $\mathrm{Var}\left[\alpha_{22}^{(1)}\right] = (5.93 \times 10^{-9})^2$, and using the residuals, we fix the marginal variance of the residual to $v_2^{(1)} = (3.80)^2$. We fix the correlation lengths using leave-one-out cross validation, testing 3000 different correlation parameter settings generated according to a Latin hypercube, and choosing the setting which minimises the log-Gaussian predictive likelihood. Using this prior specification, we pre-compute and store the moments of the update data $R_2^{(1)}$; we now have all the information that we need to compute adjusted expectations and covariances for risk values at new input



(a) (b)

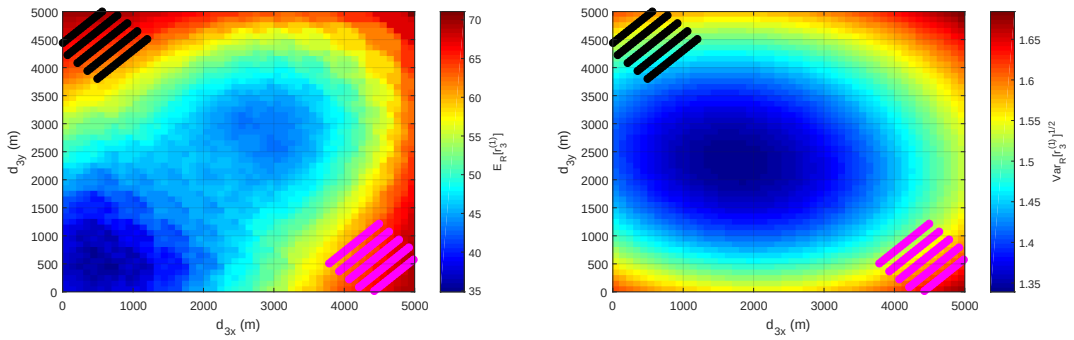
Figure 4.4: Plots of the adjusted moments for the emulator $r_2^{(1)}$. Figure 4.4(a) shows the adjusted expectations $\mathrm{E}_{R_2^{(1)}}\left[r_2^{(1)}\right]$ and Figure 4.4(b) shows the adjusted standard deviations $\mathrm{Var}_{R_2^{(1)}}\left[r_2^{(1)}\right]^{1/2}$ for a range of different (d_{2x}, d_{2y}) settings, for fixed d_1 (flight path shown as black markers), with $d_{jh} = 200$, $d_{jw} = 1000$ and $d_{jd} = 200$ for $j = 1, 2$.

settings.

Figure 4.4 plots the fitted emulator for a range of different (d_{2x}, d_{2y}) settings, for a fixed flight path design d_1 at the first stage; all predictions are made for fixed settings of $\{d_{jh}, d_{jw}, d_{jd}\}$ at stages 1 and 2.

Stage $j = 1$ Moving back to stage $j = 1$ of the calculation, our first task is again to characterise our uncertainty about the risk $s_2^{(1)}$ from an optimal design at stage 2. We adopt the same strategy as before; to reduce the computational burden of the fitting procedure, we choose to use only a single candidate minimum sample $\tilde{d}_2$ to characterise the expectation (3.4.17) and the second component of the covariance (3.4.18) for each setting of the inputs $\{z_{[1]}, w_{[1]}, d_{[1]}\}$. We approximate the first component of the covariance (3.4.18) by generating 20 samples at each of 10 input settings, and computing the variance of $\mathrm{E}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\right] + c_2\left(\tilde{d}_2\right)$ at each set of inputs. We fix $\mathrm{Var}\left[\mathrm{E}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\left[\tilde{d}_2\right]\right] + c_2\left(\tilde{d}_2\right)\right] = (2.06)^2$, the average variance across these 10 input locations, and fix all covariances between different input settings to zero.

We now proceed to fit the emulator; we use 100 risk evaluations for the initial regression, 300 for the joint regression-residual update, and 100 for model checking.



(a) (b)

Figure 4.5: Plots of the adjusted moments for the emulator $r_1^{(1)}$. Figure 4.5(a) shows the adjusted expectations $\mathrm{E}_{R_1^{(1)}}\left[r_1^{(1)}\right]$ and Figure 4.5(b) shows the adjusted standard deviations $\mathrm{Var}_{R_1^{(1)}}\left[r_1^{(1)}\right]^{1/2}$ for a range of different (d_{1x}, d_{1y}) settings. For all predictions, the other design parameters are fixed to $d_{1h} = 200$, $d_{1w} = 1000$ and $d_{1d} = 200$.

Performing the initial regression, we fix $\mathrm{E}\left[\alpha_{11}^{(1)}\right] = 7.57$ and $\mathrm{E}\left[\alpha_{12}^{(1)}\right] = 0.622$, with $\mathrm{Var}\left[\alpha_{11}^{(1)}\right] = (6.98 \times 10^{-7})^2$ and $\mathrm{Var}\left[\alpha_{12}^{(1)}\right] = (1.32 \times 10^{-8})^2$, and we use the residuals from this regression to fix $v_1^{(1)} = (3.03)^2$. For the cross validation, we test 5000 different correlation parameter settings, generated according to a Latin hypercube, and choose the setting which minimises our predictive log-Gaussian criterion. Using this prior specification, we compute and store the prior moments of $R_1^{(1)}$; we are now in a position to be able to generate risk predictions for new input settings. Figure 4.5 shows the emulator fit at this stage for a range of different (d_{1x}, d_{1y}) settings.

Stopping All of the steps of the approximate backward induction procedure 2 have now been completed for this problem; we now examine the results, and determine the course of action that we should take, as outlined in Section 3.4.10. First, we approximately identify the design which minimises the expected risk; we do this by interrogating the surface $\mathrm{E}_{R_1^{(1)}}\left[r_1^{(1)}\left[d_1\right]\right] + c_1\left(d_1\right)$ at a Latin hypercube of 2000 points, and fixing $\hat{d}_1$ to be the design which minimises the risk surface over this set. At this point, we find that $\mathrm{E}_{R_1^{(1)}}\left[r_1^{(1)}\left[\hat{d}_1\right]\right] + c_1\left(\hat{d}_1\right) = 32.26$, with $\mathrm{Var}_{R_1^{(1)}}\left[r_1^{(1)}\left[\hat{d}_1\right]\right] = (1.19)^2$. Figure 4.6 shows a sample of 100 candidate designs

generated using the procedure 3; we see from this plot the extent of the constraints imposed on the design space by the risk emulator. The range of designs which we still believe could be optimal has been strongly restricted in (d_{1x}, d_{1y}) space, and we also see some restrictions emerging in the other three input spaces.

We compare this risk with the risk $\rho_0^t = 196.34$ from an immediate decision under our prior beliefs; we are the risk from an optimally-designed future procedure is much less than the risk from an immediate decision, and so we conclude that it we were to proceed without further analysis of the risk, we would perform an experiment at $\hat{d}_1$ before considering the second experiment in the light of this outcome.

Having established what we would do if we were to continue, we compute expected value of perfect information for the risk calculation, to help us decide whether a further wave of the procedure would be beneficial. We do this by sampling 200 candidate designs, and their corresponding risks, using algorithm 3; we find that $v_1^{(1)} = 0.62$; this is around 2% of our expectation for the risk from the experiment at $\hat{d}_1$. Whether or not we would choose to perform another wave of the procedure at this point would depend on the cost of the computational resources required to do so. If this cost is small relative to the losses that we might incur by selecting a bad design, then we might pay for another wave (see the discussion in Section 3.4.10). We do not attempt to quantify the cost of running another wave in this work; we simply perform another wave to demonstrate the procedure.

As an additional diagnostic tool, we also generate risk profile plots, shown in Figure 4.6. We repeatedly simulate the sequential design procedure, by selecting a design for each experiment using the emulators fitted at this wave of the procedure (algorithm 3), comparing risks to see whether we should perform the experiment (Section 3.4.10), and then sampling the corresponding external parameters and data for the experiment if we decide to continue. The risk profile plot gives us an indication of the spread of risks that we might expect if we were to design our experiments based on the information provided by our current emulators.

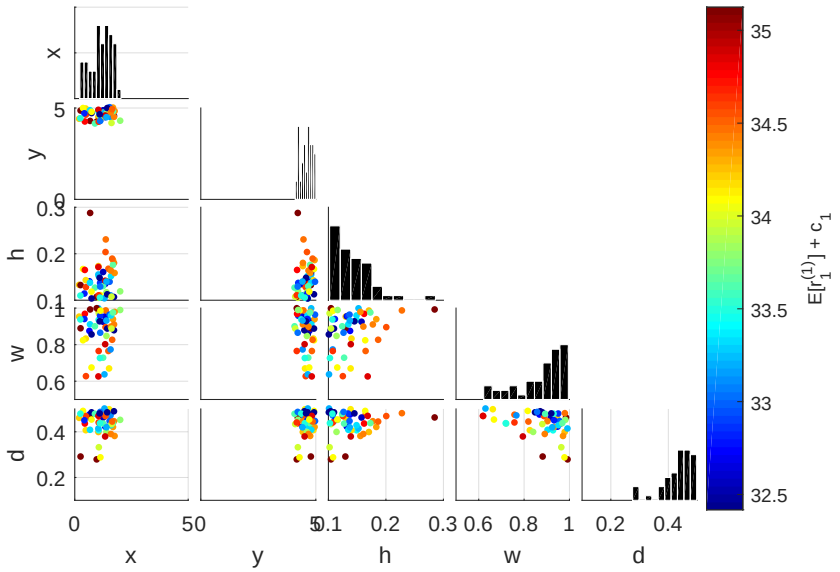
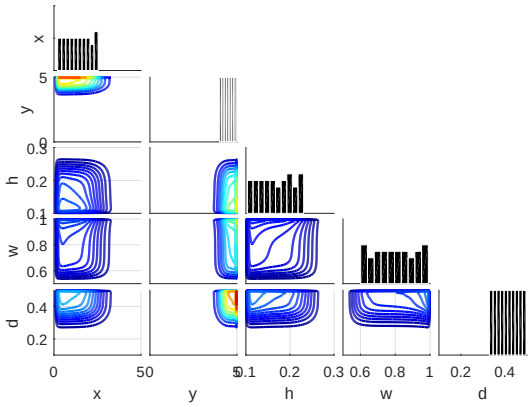


Figure 4.6: Scatter plot of 100 candidate design points $\tilde{d}_1$ generated using the algorithm 3. The colour scale indicates the expected risk $\mathrm{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\left[\tilde{d}_1\right]\right] + c_1\left(\tilde{d}_1\right)$ at each of these points.

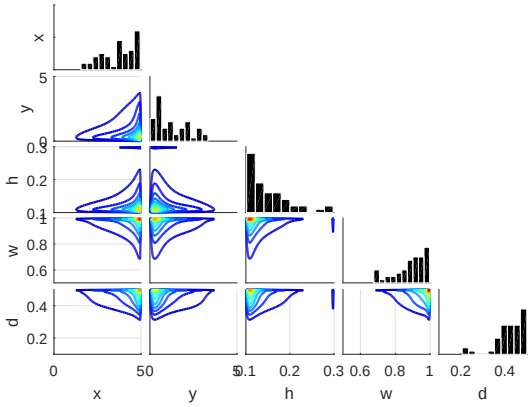
4.2.3 Second wave

At the second wave, we re-run the algorithm inside the parts of the design space which are not screened out by our emulators at the first wave. We choose points $\{z_{[j]k}, w_{[j]k}, d_{[j]k}\}$ at which to run the risk as outlined in Section 3.4.11; we choose only designs d_j which feature in the candidate design spaces defined by our emulators at the first wave, and we check all the designs that we generate using the criterion (3.4.20). Figure 4.7 shows contour plots of the densities of 100 designs $d_{[3]}$ sampled in this way; we see that the emulators at the first wave impose strong constraints on the design space at all three stages.

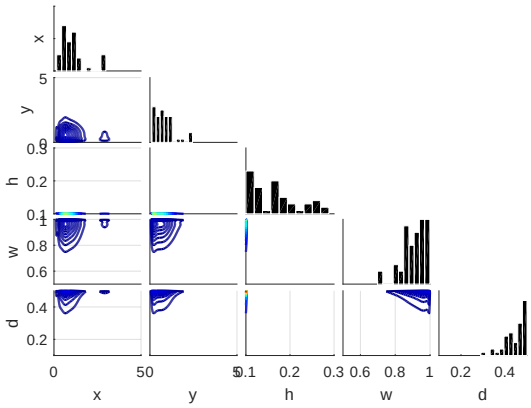
Each candidate design that we generate requires us to run the algorithm 3, which is a computationally expensive task. We see from Figure 4.6 that the designs that we generate through running this procedure are concentrated in certain regions of the design space. On this basis, instead of using the full procedure from Section 3.4.11 to generate design inputs for our second wave, we simply identify the space of design parameters for the second wave by sampling a single set of candidate designs $\{\tilde{d}_1, \tilde{d}_2, \tilde{d}_3\}$ across the three stages, and restricting designs to lie between the



(a) $j = 1$

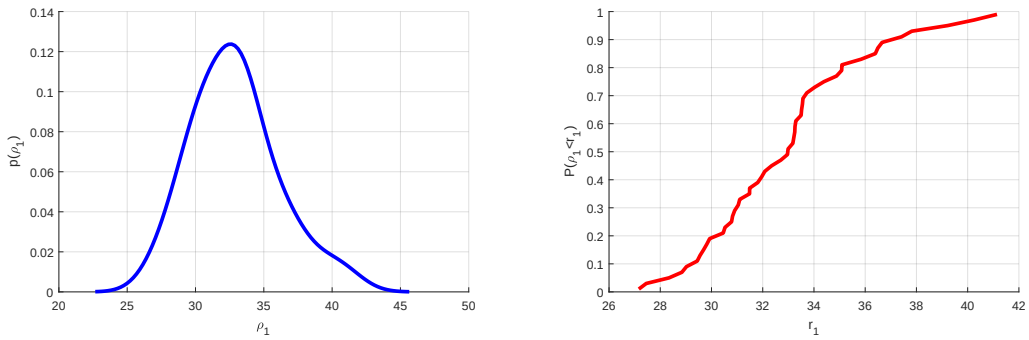


(b) $j = 2$



(c) $j = 3$

Figure 4.7: Contour plots showing the densities of a sample of 100 candidate design points $\{\hat{d}_1, \hat{d}_2, \hat{d}_3\}$ sequentially generate using the procedure 3; red contours enclose regions of high density and blue contours enclose regions of low density. All densities were generated using the 'ksdensity' function in Matlab.



(a) (b)

Figure 4.8: Risk profile plots for the first wave of the approximate backward induction procedure, generated by sampling experimental trajectories according to the emulators that we fitted at wave $i = 1$. Figure 4.8(a) shows the approximate density of risks, and Figure 4.8(b) shows the corresponding cumulative density; both plots are generated using a sample of 50 trajectories.

minimum and maximum values generated.

The basis and covariance functions that we use for all fits are the same as for the first wave, as outlined in Section 4.2.1. Below, we provide details relating to the emulator fit at each stage. As at the first wave, all fitted emulators are checked against an additional set of risk evaluations, to ensure that fewer than 5% of the new points lie outside three-standard deviation predictive error bars generated from the emulator. Additionally, as discussed in Section 3.4.4, we check all sets of risk evaluations that we generate at this wave against the emulators fitted at the first wave, again ensuring that fewer than 5% lie outside predictive error bars.

Stage $j = 3$ We first generate 200 risk evaluations for the initial regression, 500 for the joint regression-residual update, and 100 for post-fit model checking. All data at this stage is generated directly from the terminal risk ρ_3^t. Performing the initial regression, we fix $\mathrm{E}\left[\alpha_{31}^{(2)}\right] = 0.860$ and $\mathrm{E}\left[\alpha_{32}^{(2)}\right] = 0.9831$, with $\mathrm{Var}\left[\alpha_{31}^{(2)}\right] = (3.5 \times 10^{-3})^2$ and $\mathrm{Var}\left[\alpha_{32}^{(2)}\right] = (8.64 \times 10^{-5})^2$; using the regression residuals, we fix $v_3^{(2)} = (1.23)^2$.

We carry out the same cross-validation procedure, comparing 5000 different settings

of the correlation parameters generated according to Latin hypercube and selecting the setting which minimises the sum of the predictive log-Gaussian likelihoods for the left-out points. Based on this prior specification, we compute the moments of the data $R_3^{(2)}$ and store, in preparation for computing adjusted predictions at new risk input settings. This completes the emulator fit at this stage.

Stage $j = 2$ When characterising $s_3^{(2)}$, we choose the same approximations as at wave 1. The expectation (3.4.17) is fixed to the expectation $\mathrm{E}\left[\bar{r}_3^{(2)}\left[\tilde{d}_3\right]\right] + c_3\left(\tilde{d}_3\right)$ at a single candidate design, and the second term of the covariance (3.4.18) is approximated by the covariances at the same points. The first term of (3.4.18) is fixed to be constant across the input space. We fix $\mathrm{Var}\left[\mathrm{E}_{R_3^{(2)}}\left[\bar{r}_3^{(2)}\left[\tilde{d}_3\right]\right] + c_3\left(\tilde{d}_3\right)\right] = (1.05)^2$ by sampling 20 candidate designs at 10 different input locations, computing the variance of the expectation at each, and computing the average variance across these input settings; we fix all covariances between expected risks at different input locations to be zero.

We generate 50 risk evaluations for the initial regression, 200 for the joint regression-residual update, and 50 for post-fit model checking. The initial regression leads us to fix $\mathrm{E}\left[\alpha_{21}^{(2)}\right] = 12.3$ and $\mathrm{E}\left[\alpha_{22}^{(2)}\right] = 0.458$, with $\mathrm{Var}\left[\alpha_{21}^{(2)}\right] = (1.03 \times 10^{-6})^2$ and $\mathrm{Var}\left[\alpha_{22}^{(2)}\right] = (2.45 \times 10^{-8})^2$; the residuals from this regression are used to fix $v_2^{(2)} = (2.70)^2$. For this fit, our cross validation chooses the best-performing point (under the usual criterion) from a Latin hypercube of 5000 correlation parameter settings.

Stage $j = 1$ Our procedure for characterising $s_2^{(2)}$ is the same as for stage 3, and for this stage at the previous wave; in this instance, our approximation to the first term of the covariance (3.4.18) for an individual input setting is $\mathrm{Var}\left[\mathrm{E}_{R_2^{(2)}}\left[\bar{r}_2^{(2)}\left[\tilde{d}_2\right]\right] + c_2\left(\tilde{d}_2\right)\right] = (0.737)^2$, and we again set the covariances between $s_2^{(2)}$ values at different input settings to zero.

50 risk evaluations are used for the initial regression, 200 are used for the joint regression-residual update, and 50 are used for model checking after the fit. After the initial regression, we fix $\mathrm{E}\left[\alpha_{11}^{(2)}\right] = 2.95$ and $\mathrm{E}\left[\alpha_{12}^{(2)}\right] = 0.766$, with $\mathrm{Var}\left[\alpha_{11}^{(2)}\right] = (2.76 \times 10^{-6})^2$ and $\mathrm{Var}\left[\alpha_{12}^{(2)}\right] = (8.37 \times 10^{-8})^2$; the residuals from this regression are

used to fix $v_2^{(2)} = (0.974)^2$. For the cross-validation, we choose the best-performing point from a Latin hypercube of 5000 correlation parameters settings.

Stopping After the second wave of analysis, we again stop and determine what we should now do, as outlined in Section 3.4.10. First, we approximately identify the design setting which minimises the expected risk $\mathrm{E}\left[\bar{r}_1^{(2)}\left[d_1\right]\right] + c_1\left(d_1\right)$; we do this by interrogating the risk at a Latin hypercube of 2000 designs and fixing $\hat{d}_1$ to be the design from this set with the lowest risk. At this point, our expectation for the risk is $\mathrm{E}\left[\bar{r}_1^{(2)}\left[\hat{d}_1\right]\right] + c_1\left(\hat{d}_1\right) = 29.21$, with $\mathrm{Var}\left[\bar{r}_1^{(2)}\left[\hat{d}_1\right]\right] = (0.35)^2$. Figure 4.9 shows a set of candidate designs sampled from the emulator $\bar{r}_1^{(2)}$ using the procedure 3; we see that this second wave has resulted in a further, but more modest, reduction in the range of designs which feature in our candidate design space.

We again compute the EVPI for the risk calculation, to aid us in making a decision about whether we should run another wave of analysis. Again, we approximate the second term of equation (3.4.19) by sampling 200 candidate designs as in algorithm 3 and using the corresponding risks. We find that $v_1^{(2)} = 0.04$; this is now only about 0.14% of the risk from further experimentation. This reduced value of $v_1^{(2)}$, and the general reduction in our uncertainty about the risk, indicate that the second wave has done much to reduce our uncertainty about the risk over this region of the design space. As discussed in Section 3.4.10, whether or not we would proceed with another wave of analysis would depend on the relative costs of the resources required to do so. Clearly, though, after this second wave, it is less likely that we would choose to run another, since the maximum improvement that we could achieve ($v_1^{(2)}$) is now much smaller relative to the experimental risk. Additionally, since comparing Figures 4.6 and 4.9 indicates that the reduction in the size of the candidate design space has been more modest at this wave, we might expect that the improvement in the accuracy of our emulators over this region of the design space will not increase much further at wave 3, and therefore that $v_1^{(3)}$ might not be much less than $v_1^{(2)}$.

Figure 4.10 shows risk profile plots for the emulators fitted at this wave; as discussed in Section 4.2.2, these are generated by forward-sampling 50 experimental procedures, selecting designs according to procedure 3 and comparing experimental

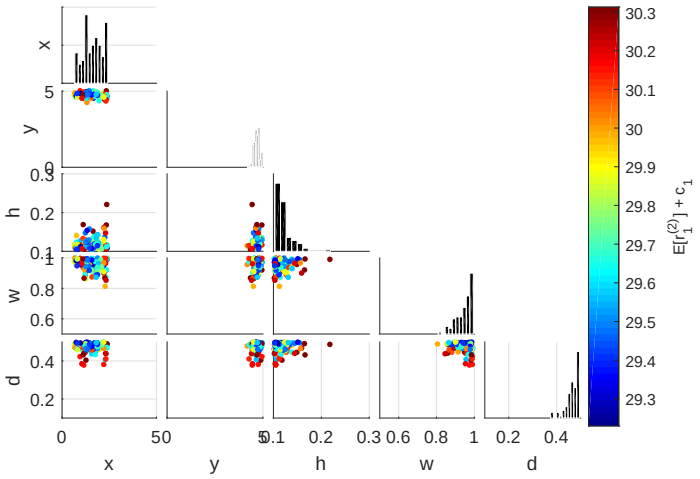


Figure 4.9: Scatter plot of candidate designs $\tilde{d}_1$ generated using algorithm 3. The colour scale indicates the expected risk $\mathrm{E}\left[\bar{r}_1^{(2)}\left[\tilde{d}_1\right]\right] + c_1\left(\tilde{d}_1\right)$ at each point.

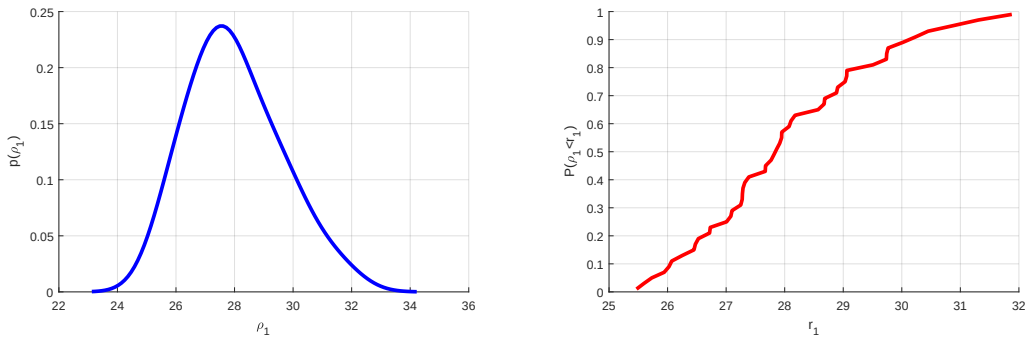
risks from our emulators with terminal risks in order to choose between stopping and continuing. We see by comparing Figure 4.10 with Figure 4.8 that after this wave, the distribution of risks that we may obtain by designing the experiments using our emulators has a much lower spread.

4.3 Discussion

Having run the algorithm for two different problems, we discuss our results; in particular, we consider the feasibility of running the algorithm for more complex models and loss functions in Section 4.3.1, and we consider further work which could be done to improve the approximation procedure's efficiency and accuracy in Section 4.3.2.

4.3.1 Practical difficulties

In Section 4.1, the approximation procedure is used for the analysis of a 3 stage problem in which the forward model has a linear dependence on the parameters of interest; some aspects of the approximation will scale well to the analysis of more complex problems, but there are also aspects which may require further development before application to harder problems will be computationally feasible.



(a) (b)

Figure 4.10: Risk profile plots for the emulators fitted at the second wave of analysis, generated by sampling trajectories of the experimental procedure, selecting designs as outlined in algorithm 3. Figure 4.10(a) shows the density of sampled risks, and Figure 4.10(b) shows the corresponding cumulative density. Both plots are generated using a sample of 50 risks.

High-dimensional design spaces Emulators are useful tools for representing the output of complex computer simulators which have long run-times (see, for example, Williamson et al. [2013], Vernon et al. [2010]); however, they are no magic bullet, and it is very easy to encounter difficulties when trying to fit such a model to represent a model which is very variable across its high-dimensional input space. The atmospheric dispersion modelling problem encountered in the previous section required us to choose a 6-dimensional design parameter at each of three stages, but it is simple to imagine a problem in which a much higher number of parameters must be specified in order to determine a particular experiment.

When it is feasible to implement, the strategy of using the modified terminal risk as a basis function is a powerful tool in enabling us to approximate risk functions well in high-dimensional input spaces; even if we can construct an accurate emulator for the risk though, the problem of locating the minimum of the risk function also increases in difficulty as the dimensionality of the design space increases, and this has the potential to be a much more difficult hurdle to overcome.

Non-linear dependence on parameters If the dependence of the model for the system on the parameters q (about which we wish to learn and make decisions) is non-linear and complex, the usual problems which we would encounter in this situation are magnified by the need to repeatedly evaluate our posterior beliefs about q for different designs, external parameters and data sets. We could approach such a problem by representing the complex deterministic parts of the model using an emulator (as in Section 2.2, in the usual way for a computer simulator), and then using the calculations presented in Sections 2.4.2 and 2.4.3 for uncertainty propagation and adjustment of beliefs about inputs; even if we can implement this efficiently, though, each inference is likely to be much slower than the inference in the example above, which may affect our ability to use the terminal risk as a mean function.

More complex loss functions In both of the examples considered above, the loss function used is simple to evaluate using our posterior (or adjusted) beliefs about the model parameters; for the weighted quadratic loss, the optimal terminal decision can be found exactly in all cases, as can the corresponding terminal risk. Such loss functions are common in situations where we are unwilling to make a more detailed specification of our decision problem, but in cases where the loss is a more complex function, we will need to take more care in our implementation.

It may be the case that the loss function for a given problem is sufficiently complex that it requires emulation in its own right. If this is the case, and we design our emulator carefully, then we can compute the expected loss as a function of a_j for any stage by integrating the emulator with respect to $p\left(q|z_{[j]}, w_{[j]}, d_{[j]}\right)$. However, if the loss is complex enough to need emulation, then it is unlikely that the optimal decision a_j^* will be available in closed-form, and so we will need to implement some form of numerical procedure (e.g. sampling using a Latin hypercube, as in Section 3.4.8, or numerically optimising the mean surface) in order to find the terminal risk. This has the potential to represent a large extra computational burden, particularly when the terminal risk is to be used as part of the basis for the risk emulator.

4.3.2 Further work

Further improvements to the algorithm presented in Section 3.4 will be focussed on improving its ability to handle high-dimensional and highly variable risks. One of the least well-developed aspects of the algorithm at the moment is the minimum sampling procedure from Section 3.4.8; this is because characterising the behaviour of the minimum of a stochastic function is a difficult problem in general. One possible solution to this would be to change our formulation of the sequential problem slightly, so that the risks which we compute retain their dependence on the design parameters from all stages; if we were to do this, we would obtain a risk emulator $\bar{r}_1^{(i)}\left[d_{[n]}\right]$ at the first stage which was a function of the design parameters from all stages, representing the risk from the sequential decision procedure in which, at each stage, we have the option of carrying out an experiment at pre-specified design parameter d_{j+1}, or making an immediate terminal decision a_j^*.

If we were to build the emulator in such a way, then the need for an approximate minimum sampling algorithm such as the one outlined in Section 3.4.8 would be eliminated for all stages except the first, since we would be able to sample all risks from known design settings exactly; this would, hopefully, improve the level of systematic risk variability that we could explain using our emulators. An issue with fitting the emulator in this way would be the increase in the dimensionality of the input space for the emulators at stages $(n-1), \ldots, 1$, and so if carrying out the analysis in this way, we must assess whether the increased effort required to fit emulators with higher-dimensional input spaces translates into a suitably improved representation of the risk function. In any case, we would still need an algorithm along the lines of the one in Section 3.4.8 in order to assess the likely location of the minimum risk at stage $j = 1$.

Chapter 5

Design for developing models

5.1 Model development: Reification

The development process for a model for a system often involves a number of simplifications and compromises:

- in the atmospheric modelling example of Hirst et al. [2013], it is extremely clear that the Gaussian plume model does not capture the true behaviour of the gas under given atmospheric conditions. The real atmosphere is considerably more turbulent than the Gaussian plume model allows, and frequently, changes in the wind over long distances cause systematic discrepancies between the predictions of the plume and the observed concentration (see, for example, Figure 1.1);

- climate models are abstractions of extremely large and complex natural systems, where even if all of the processes were fully understood, inclusion of all of these in the model would be computationally infeasible. Common simplifications in such models include the solution of governing equations on extremely coarse grids which necessarily neglects processes on length scales smaller than the chosen mesh (as in the example discussed in Goldstein and Rougier [2004]), or the introduction of a highly idealised representation of the system (as in the compartmental representation of the Atlantic ocean used by Zickfeld et al. [2004] in their model).

It is generally the case that the scientists that develop these models acquire knowledge during the development process about how they might be improved so as to better describe the behaviour of processes in the true system. For example:

- more accurate models of gas transport processes are well understood (as described in chapter 1), but are not implemented in the analysis by Hirst et. al. because of the extra computer power that would be required and the lack of availability of high-resolution atmospheric information;

- in both of the climate examples described above, it is simple to imagine (though potentially complex to implement) ways in which the model might be improved; in the first instance, we could simply solve the equations on a finer grid, or introduce alternative representations of the sub-grid-scale processes, and for the Atlantic model, we might introduce additional compartments, or refine the model output by introducing differential equations which represent other processes within the existing compartments. Generally within climate models, it is always possible to introduce representations of additional physical processes which would bring the model closer to reality.

Where the modellers have such knowledge about how the model might better represent the system, inferences and predictions about the system made without taking account of these will not be consistent with their current beliefs. One option is to continue development of the model; however, in doing this, it is highly likely that during this additional development, new ideas about further future improvements will be generated, rendering the new model inadequate as well (while incurring additional development costs). It would be better, therefore, to model the system using a framework which is capable of incorporating such beliefs without the need to actually build the future models that we postulate.

A consistent approach to handling such expert judgement has been developed by Goldstein and Rougier [2009]: in the situation where an emulator is used as a statistical representation of a simulator, we can handle beliefs about likely future improvements to the simulator or to the underlying model by using a multi-level framework, introducing additional components which represent the effects of future

developments and specifying relationships between existing and new processes across levels.

5.1.1 Reifying principle

As discussed in Section 2.4.1, the best input assumption is a common technique for linking together simulators for a system with the true, underlying values of particular system properties: we assume that there is a particular setting a^* for certain inputs such that if we were to evaluate the simulator at this point, this run would provide all of the information available from the simulator about the system, and no further evaluations would be necessary. In the situation where we are modelling simulator improvements, making such an assumption for every development stage of the simulator would be impractical: the best input settings at all stages would necessarily be strongly correlated (it is difficult to imagine a situation in which knowledge of the best input at one development stage would tell us nothing about likely values of the best input at the next stage, where the model has changed only slightly), and it would be very difficult to get an expert to specify such a correlation structure.

Indeed, where we have beliefs about how our current simulator might improve in the future, it is not even clear what we mean when we speak about a 'best input' assumption for an individual simulator; if we believe that the simulators are correlated across their input spaces, then an evaluation of any single simulator is informative for all of the evaluations that we could make on any of the others, and so the situation in which a single run of such a simulator could provide us with all of the information available in the model about the system (see the discussion of the best input assumption in Section 2.3.1) cannot arise. In the above paragraph, therefore, what we actually intend to say is that the best input settings would be correlated across the stages of development, were we to specify correlations between the simulators and then attempt to link each of the simulators to the system.

To overcome these difficulties, and to develop a framework which handles our beliefs coherently, Goldstein and Rougier [2009] suggest the adoption of the reifying principle, under which we imagine the best possible simulator that we might construct

for the system, and then assume the following:

Reifying principle: The reified simulator separates our actual simulators from the underlying system. Our actual simulators are informative for the underlying system because they are informative for the reified simulator.

Under this principle, the best input assumption is applied only to the reified simulator; the other simulators that we might construct are then informative for the system only because they are informative for the reified simulator, and not because they are linked directly to the system in any way.
The linking of the different system simulators is achieved by using an emulator as a description of each simulator; we use our judgements about future simulator behaviour to specify relationships between the basis coefficient and residual components of the emulators at different stages. We link the reified simulator to the system by making a best input assumption, and by specifying our beliefs about the discrepancy between the reified simulator (evaluated at its best input) and the system. For simplicity, in this thesis, we assume that models are only directly linked together if they correspond to successive development stages, and we do not allow for the possibility that we may choose multiple 'development directions' at any given stage. In practice, though, it is common that multiple different improvements to the model might be investigated simultaneously, by different parties. This point is discussed further in Section 5.8.

5.2 Model structure

We specify our beliefs about how our model may evolve in the future within the framework outlined by Goldstein and Rougier [2009]. We assume that we can envisage constructing n_s future simulators; simulators $k = 1, \ldots, (n_s - 1)$ are indexed as $f^{(k)}(.)$, and the reified simulator is denoted by $f^*(.)$ and is the $n_s{}^{\text{th}}$ model. We use an emulator to approximate the behaviour of each of these simulators as a function of inputs θ, and assume that each of these emulators has the standard form (see Section 2.2.1); we adopt the summation convention again, with the additional

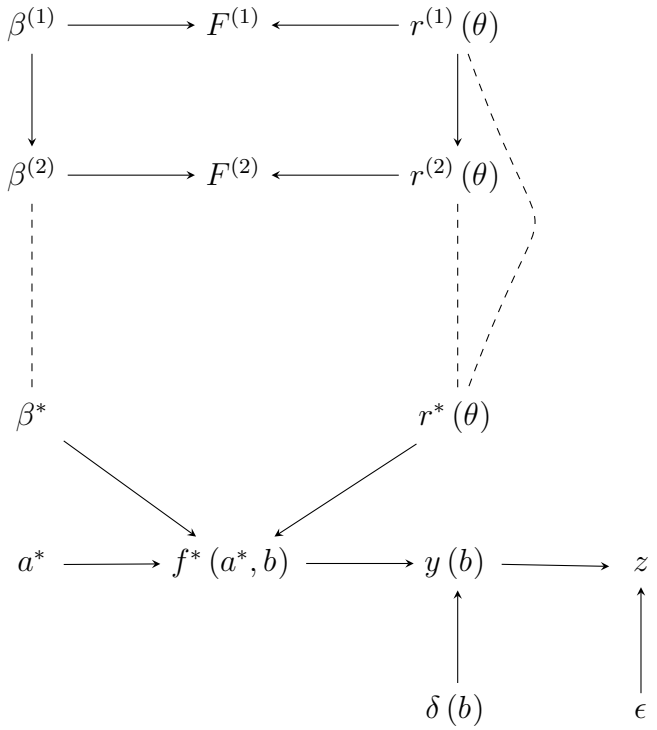


Figure 5.1: DAG displaying the structure of the reified model.

requirement that superscript indices are not summed over

$$\begin{aligned} f_i^{(k)}(\theta) &= g_j^{(k)}(\theta)\,\beta_{ij}^{(k)} + r_i^{(k)}(\theta) \\ f_i^*(\theta) &= g_j^*(\theta)\,\beta_{ij}^* + r_i^*(\theta) \ . \end{aligned}$$

Note that while the set of input parameters required to run each model may vary, for simplicity, we assume that θ is the set of inputs that is sufficient to run all models, including both system and model inputs.

In general, to make a full second-order prior specification for this model, we would need to specify expectations and variances for each $\beta^{(k)}$ and covariances between each pair $\{\beta^{(k)}, \beta^{(l)}\}$ and $\{r^{(k)}(.), r^{(l)}(.)\}$: however, as discussed in Section 5.1, we want our model to have the property that each improvement to the simulator separates all previous versions from the system, with all simulators linked to the system through the reified simulator (which we will never actually construct).

We therefore choose the following set of relationships between the model components at the different development stages: for the basis coefficients, we only require that $\beta^{(j)} \perp\!\!\!\perp \beta^{(l)} | \beta^{(k)}$ for $j < k < l$, so that knowledge of the set of coefficients at

development level k is sufficient for inferences about the coefficients at all higher-numbered levels $l > k$ given the set $\{\beta^{(1)}, \ldots, \beta^{(k)}\}$ of coefficients up to level k. For the residual processes, the situation is more complex: only knowledge of a process for every setting of its inputs is enough to separate the residual processes across levels in the same way. We assume the following, simple form for our relationships between the simulators

$$r_i^{(k)}(\theta) = \gamma_{ij}^{(k)}(\theta)\, r_j^{(k-1)}(\theta) + \nu_i^{(k)}(\theta)$$

where $\gamma_{ij}^{(k)}(.)$ is a known weight function determining the contribution of the residual function $r_j^{(k-1)}(.)$ to the residual function $r_i^{(k)}(.)$ at the next development stage for each setting of the inputs, and $\nu_i^{(k)}(.)$ is an additional stochastic term which is assumed to be a-priori uncorrelated with $r_j^{(k-1)}(.)$. Under this specification, if we use the same input set for the simulator runs $F^{(k)}$ at each level, then we recover the separation property for the residual between levels of the model.

This model structure is depicted in the DAG in Figure 5.1. While it may not always be easy to directly specify a correlation structure between the $\beta^{(k)}$ at adjacent levels, these can derived from any set of linear relationships between components, making this an extremely flexible general specification to work with.

5.3 Calculations

We now consider how we do calculations using this model: first, in Section 5.3.1, we use the structure of the model to turn the limited prior specification outlined in Section 5.2 into a full joint prior specification for all components of the model; then, in Section 5.3.2, we use this prior specification to compute corresponding prior beliefs about the simulator runs that we will obtain, and use these runs to adjust beliefs about the rest of the model; finally, in Section 5.3.3, we use these adjusted beliefs to propagate uncertainty on a subset of the inputs onto uncertainty about the system.

5.3.1 Full joint prior

The graphical representation of the model depicted in Figure 5.1 can be used as a framework within which we can compute the full joint prior specification. Before the simulator runs $\{F^{(k)}\}$ have been observed, the basis coefficients and the residuals are assumed to be independent, and so we consider the prior specification for each separately.

In some situations, it may be that there is very little prior information about the behaviour of any of the simulators until one has been built and run; in these situations, it may be preferable to construct the first simulator and then emulate this, before using this emulator to posit relationships between future versions. If we carry out our analysis in this order, the observation of the initial runs will induce a covariance between the coefficients $\beta^{(k)}$ and residuals $r^{(k)}(.)$ in the resulting prior specification. While this would not introduce any significant extra complexity into the calculations that we must perform, this is not how we will proceed in our example, and so we do not consider this issue further here.

Basis coefficients Our prior specification consists of expectations and variances $\mathrm{E}\left[\beta^{(k)}\right]$, $\mathrm{Var}\left[\beta^{(k)}\right]$ for each set of coefficients, and covariances $\mathrm{Cov}\left[\beta^{(k)}, \beta^{(k-1)}\right]$ between adjacent coefficient sets. Because of the belief separation structure imposed on the model, we can use the graph to compute the covariance between any two sets of coefficients: because the $\beta^{(k)}$ are separated from all $\beta^{(l)}$, $l < k$ by $\beta^{(k-1)}$, covariances between these sets of coefficients can be computed as

$$\mathrm{Cov}\left[\beta_{ij}^{(k)}, \beta_{pq}^{(l)}\right] = \mathrm{Cov}\left[\beta_{ij}^{(k)}, \beta_{rs}^{(k-1)}\right] \mathrm{Var}\left[\beta^{(k-1)}\right]^{-1}_{rstu} \mathrm{Cov}\left[\beta_{tu}^{(k-1)}, \beta_{pq}^{(l)}\right] .$$

Residuals For the residual processes, the relationship between stages can be used to compute the general covariance structure. For two evaluations of the residual process at the same development level, the covariance between them is defined recursively

$$\begin{aligned}\mathrm{Cov}\left[r_i^{(k)}(\theta), r_j^{(k)}(\theta')\right] =& \gamma_{ip}^{(k)}(\theta)\, \mathrm{Cov}\left[r_p^{(k-1)}(\theta), r_q^{(k-1)}(\theta')\right] \gamma_{jq}^{(k)}(\theta') \\ &+ \mathrm{Cov}\left[\nu_i^{(k)}(\theta), \nu_j^{(k)}(\theta')\right] .\end{aligned}$$

Across stages, we pick up additional factors of the scaling functions: for $k > l$, we have

$$\mathrm{Cov}\left[r_i^{(k)}(\theta), r_j^{(l)}(\theta')\right] = \gamma_{ip}^{(k)}(\theta)\,\gamma_{pq}^{(k-1)}(\theta)\,\ldots\,\gamma_{vw}^{(l+1)}(\theta)\,\mathrm{Cov}\left[r_w^{(l)}(\theta), r_j^{(l)}(\theta')\right] .$$

5.3.2 Adjustment

Using the prior specification outlined in Section 5.3.1, we now consider the effect on our beliefs of observing runs on some of the simulators. First, we derive our prior beliefs about the simulator runs; then, we observe the runs, and use these to adjust our prior beliefs about all simulator components; finally, we consider the effect of this adjustment on our beliefs about the behaviour of the system.

Moments of the simulator runs Having derived the full joint prior specification for the simulators, we can use this to calculate the moments of the simulator data that we will use to adjust our beliefs. To simplify the notation for the adjustment, we lower the index for the level, using F_{ijk} to denote the i^{th} output of the k^{th} simulator evaluated at input setting θ_j

$$F_{ijk} = f_i^{(k)}(\theta_j) .$$

Recall that superscript indices are not summed over. The prior moments of these values can be derived directly from the prior specification: the means are

$$\mathrm{E}\left[F_{ijk}\right] = \mathrm{E}\left[\beta_{il}^{(k)}\right] G_{ljk}$$

where the elements of the design matrix are

$$G_{ljk} = g_l^{(k)}(\theta_j)$$

and the covariances between values can be computed as

$$\begin{aligned}\mathrm{Cov}\left[F_{ijk}, F_{pqr}\right] = \; & G_{ljk}\mathrm{Cov}\left[\beta_{il}^{(k)}, \beta_{ps}^{(r)}\right] G_{sqr} \\ & + \mathrm{Cov}\left[r_i^{(k)}(\theta_j), r_p^{(r)}(\theta_q)\right] .\end{aligned}$$

Additionally, in order to perform the update, we will need the following covariances, which we define here for notational simplicity

$$\begin{aligned}\mathrm{Cov}\left[\beta_{ij}^{(k)}, F_{pqr}\right] &= \mathrm{Cov}\left[\beta_{ij}^{(k)}, \beta_{ps}^{(r)}\right] G_{sqr} \\ \mathrm{Cov}\left[r_i^{(k)}(\theta), F_{pqr}\right] &= \mathrm{Cov}\left[r_i^{(k)}(\theta), r_p^{(r)}(\theta_q)\right] .\end{aligned}$$

Using these moments of the observed simulator runs, we first adjust the prior moments of the simulator at all stages; later, using these adjusted moments, we will incorporate the discrepancy in order to compute our corresponding beliefs about the system.

Adjusted moments of the simulators We now use the output from the simulators that we have run in conjunction with the model structure outlined in Section 5.3.2 to derive our adjusted beliefs about each of the simulators. Using the relevant properties of adjusted expectations and covariances, the adjusted expectation for the simulator $f^{(k)}(.)$ evaluated at a new input θ is

$$\mathrm{E}_F\left[f_i^{(k)}(\theta)\right] = \mathrm{E}_F\left[\beta_{ij}^{(k)}\right] g_j^{(k)}(\theta) + \mathrm{E}_F\left[r_i^{(k)}(\theta)\right] \tag{5.3.1}$$

and the adjusted covariances between simulators $f^{(k)}(.)$ and $f^{(l)}(.)$ at input settings θ and θ' are

$$\begin{aligned}\mathrm{Cov}_F\left[f_i^{(k)}(\theta), f_j^{(l)}(\theta')\right] = \; & g_p^{(k)}(\theta)\,\mathrm{Cov}_F\left[\beta_{ip}^{(k)}, \beta_{jq}^{(l)}\right] g_q^{(l)}(\theta') \\ & + \mathrm{Cov}_F\left[r_i^{(k)}(\theta), \beta_{jq}^{(l)}\right] g_q^{(l)}(\theta') \\ & + g_p^{(k)}(\theta)\,\mathrm{Cov}_F\left[\beta_{ip}^{(k)}, r_j^{(l)}(\theta')\right] \\ & + \mathrm{Cov}_F\left[r_i^{(k)}(\theta), r_j^{(l)}(\theta')\right] .\end{aligned} \tag{5.3.2}$$

In order to evaluate the expressions (5.3.1) and (5.3.2), we must compute the relevant adjusted moments of the components; these are computed in the same way as in Section 2.2.2, taking into account the extra index over simulator level. The adjusted expectations of the coefficients and the residuals are

$$\begin{aligned}\mathrm{E}_F\left[\beta_{ip}^{(k)}\right] &= \mathrm{E}\left[\beta_{ip}^{(k)}\right] + \mathrm{Cov}\left[\beta_{ip}^{(k)}, F_{klm}\right] \mathrm{Var}\left[F\right]^{-1}_{klmrst} \left[F_{rst} - \mathrm{E}\left[F_{rst}\right]\right] \\ \mathrm{E}_F\left[r_i^{(k)}(\theta)\right] &= \mathrm{Cov}\left[r_i^{(k)}(\theta), F_{klm}\right] \mathrm{Var}\left[F\right]^{-1}_{klmrst} \left[F_{rst} - \mathrm{E}\left[F_{rst}\right]\right]\end{aligned}$$

and the adjusted covariances are

$$\begin{aligned}
\mathrm{Cov}_F\left[\beta_{ip}^{(k)}, \beta_{jq}^{(l)}\right] &= \mathrm{Cov}\left[\beta_{ip}^{(k)}, \beta_{jq}^{(l)}\right] \\
&\quad - \mathrm{Cov}\left[\beta_{ip}^{(k)}, F_{rst}\right] \mathrm{Var}\left[F\right]^{-1}_{rstuvw} \mathrm{Cov}\left[F_{uvw}, \beta_{jq}^{(l)}\right] \\
\mathrm{Cov}_F\left[r_i^{(k)}\left(\theta\right), r_j^{(l)}\left(\theta'\right)\right] &= \mathrm{Cov}\left[r_i^{(k)}\left(\theta\right), r_j^{(l)}\left(\theta'\right)\right] \\
&\quad - \mathrm{Cov}\left[r_i^{(k)}\left(\theta\right), F_{rst}\right] \mathrm{Var}\left[F\right]^{-1}_{rstuvw} \mathrm{Cov}\left[F_{uvw}, r_j^{(l)}\left(\theta'\right)\right] \\
\mathrm{Cov}_F\left[r_i^{(k)}\left(\theta\right), \beta_{jq}^{(l)}\right] &= -\mathrm{Cov}\left[r_i^{(k)}\left(\theta\right), F_{rst}\right] \mathrm{Var}\left[F\right]^{-1}_{rstuvw} \mathrm{Cov}\left[F_{uvw}, \beta_{jq}^{(l)}\right] \ .
\end{aligned}$$

Beliefs about the system Using our adjusted beliefs about the reified simulator, we compute our beliefs about the system itself by incorporating uncertainty about the best input setting and discrepancy. As in Section 2.4.2, we split our input set $\theta = \{a, b\}$ into simulator inputs a and system inputs b, and we adopt the best input assumption, where our beliefs about a^* are summarised by the probability distribution $p\left(a^*\right)$. Under this assumption, the moments of the system are

$$\mathrm{E}\left[y_i\left(b\right)\right] = \mathrm{E}\left[\hat{f}_i^*\left(b\right)\right] + \mathrm{E}\left[\delta_i\left(b\right)\right] \tag{5.3.3}$$

$$\mathrm{Cov}\left[y_i\left(b\right), y_j\left(b'\right)\right] = \mathrm{Cov}\left[\hat{f}_i^*\left(b\right), \hat{f}_j^*\left(b'\right)\right] + \mathrm{Cov}\left[\delta_i\left(b\right), \delta_j\left(b'\right)\right] \tag{5.3.4}$$

where we have specified that the reified simulator and the discrepancy $\delta\left(.\right)$ are a priori independent, and that

$$\hat{f}_i^*\left(b\right) = f_i^*\left(a^*, b\right) \ .$$

The moments of $\hat{f}_i^*\left(.\right)$ are computed in Section 5.3.3. Finally, we assume that the $z_{ij} = y_i\left(b_j\right) + \epsilon_{ij}$ are noise-corrupted observations of the system; incorporating an uncertainty specification for the measurement error ϵ, the moments of the data are

$$\begin{aligned}
\mathrm{E}\left[z_{ij}\right] &= \mathrm{E}\left[y_i\left(b_j\right)\right] \\
\mathrm{Cov}\left[z_{ij}, z_{kl}\right] &= \mathrm{Cov}\left[y_i\left(b_j\right), y_k\left(b_l\right)\right] + \mathrm{Cov}\left[\epsilon_{ij}, \epsilon_{kl}\right]
\end{aligned}$$

where, as usual, we have assumed that the measurement error is independent of the system and its inputs.

5.3.3 Propagation and Calibration

Uncertainty propagation As with the standard emulator outlined in chapter 2, we want to be able to propagate beliefs about the best input through our emulator representation of the reified simulator, so that we can evaluate the moments (5.3.3) and (5.3.4) of the system; as in Section 2.4.2, the expectation of the reified simulator at its best input is found as follows

$$\mathrm{E}\left[\hat{f}_i^*(b)\right] = \mathrm{E}\left[\mathrm{E}_F\left[f_i^*(a^*,b)\,|a^*\right]\right]$$

where the outer expectation is taken with respect to $p(a^*)$. Inserting the expression for the adjusted expectation, we obtain

$$\begin{aligned}\mathrm{E}\left[\hat{f}_i^*(b)\right] &= \mathrm{E}_F\left[\beta_{ij}^*\right]\int g_j^*(a^*,b)\,p(a^*)\,da^* \\ &\quad + W_{klm}\int \mathrm{Cov}\left[r_i^*(a^*,b),F_{klm}\right]p(a^*)\,da^* \end{aligned} \tag{5.3.5}$$

where $W_{klm} = \mathrm{Var}[F]^{-1}_{klmrst}\left[F_{rst} - \mathrm{E}\left[F_{rst}\right]\right]$. The covariance is again found using the law of total covariance

$$\begin{aligned}\mathrm{Cov}\left[\hat{f}_i^*(b),\hat{f}_k^*(b')\right] &= \mathrm{E}\left[\mathrm{Cov}_F\left[f_i^*(a^*,b),f_k^*(a^*,b')\,|a^*\right]\right] \\ &\quad + \mathrm{Cov}\left[\mathrm{E}_F\left[f_i^*(a^*,b)\,|a^*\right],\mathrm{E}_F\left[f_k^*(a^*,b')\,|a^*\right]\right]\end{aligned} \tag{5.3.6}$$

where again, the outer expectations and covariances are taken with respect to $p(a^*)$. Considering the second component of (5.3.6) first, we must evaluate

$$\begin{aligned}&\mathrm{Cov}\left[\mathrm{E}_F\left[f_i^*(a^*,b)\,|a^*\right],\mathrm{E}_F\left[f_k^*(a^*,b')\,|a^*\right]\right] = \\ &\qquad \mathrm{E}_F\left[\beta_{ij}^*\right]\mathrm{E}_F\left[\beta_{kl}^*\right]\int g_j^*(a^*,b)\,g_l^*(a^*,b')\,p(a^*)\,da^* \\ &\quad + \mathrm{E}_F\left[\beta_{ij}^*\right]W_{pqr}\int g_j^*(a^*,b)\,\mathrm{Cov}\left[r_k^*(a^*,b'),F_{pqr}\right]p(a^*)\,da^* \\ &\quad + W_{pqr}\mathrm{E}_F\left[\beta_{kl}^*\right]\int \mathrm{Cov}\left[r_i^*(a^*,b),F_{pqr}\right]g_l^*(a^*,b')\,p(a^*)\,da^* \\ &\quad + W_{lmn}W_{pqr}\int \mathrm{Cov}\left[r_i^*(a^*,b),F_{lmn}\right]\mathrm{Cov}\left[r_k^*(a^*,b'),F_{pqr}\right]p(a^*)\,da^* \\ &\qquad - \mathrm{E}\left[\hat{f}_i^*(b)\right]\mathrm{E}\left[\hat{f}_k^*(b')\right]\end{aligned} \tag{5.3.7}$$

and for the first, we have

$$\begin{aligned}
\mathrm{E}\left[\mathrm{Cov}_F\left[f_i^*\left(a^*,b\right),f_k^*\left(a^*,b'\right)|a^*\right]\right] = \\
&\mathrm{Cov}_F\left[\beta_{ij}^*,\beta_{kl}^*\right]\int g_j^*\left(a^*,b\right)g_l^*\left(a^*,b'\right)p\left(a^*\right)da^* \\
-\mathrm{Var}\left[F\right]^{-1}_{pqrstu}&\mathrm{Cov}\left[F_{stu},\beta_{kl}^*\right]\int \mathrm{Cov}\left[r_i^*\left(a^*,b\right),F_{pqr}\right]g_l^*\left(a^*,b'\right)p\left(a^*\right)da^* \\
-\mathrm{Cov}\left[\beta_{ij}^*,F_{pqr}\right]&\mathrm{Var}\left[F\right]^{-1}_{pqrstu}\int g_j^*\left(a^*,b\right)\mathrm{Cov}\left[r_k^*\left(a^*,b'\right),F_{stu}\right]p\left(a^*\right)da^* \\
&+\int \mathrm{Cov}\left[r_i^*\left(a^*,b\right),r_k^*\left(a^*,b'\right)\right]p\left(a^*\right)da^* \\
-\mathrm{Var}\left[F\right]^{-1}_{pqrstu}&\int \mathrm{Cov}\left[r_i^*\left(a^*,b\right),F_{pqr}\right]\mathrm{Cov}\left[r_k^*\left(a^*,b'\right),F_{stu}\right]p\left(a^*\right)da^* \ . \quad (5.3.8)
\end{aligned}$$

Calibration Also as before (Section 2.4.3), we can use the data observed on the system to learn about a^*; the adjusted moments are computed using the usual Bayes linear adjustment equations

$$\begin{aligned}
\mathrm{E}_z\left[a_i^*\right] &= \mathrm{E}\left[a_i^*\right] + \mathrm{Cov}\left[a_i^*,z_{kl}\right]\mathrm{Var}\left[z\right]^{-1}_{klpq}\left[z_{pq}-\mathrm{E}\left[z_{pq}\right]\right] \\
\mathrm{Cov}_z\left[a_i^*,a_j^*\right] &= \mathrm{Cov}\left[a_i^*,a_j^*\right] - \mathrm{Cov}\left[a_i^*,z_{kl}\right]\mathrm{Var}\left[z\right]^{-1}_{klpq}\mathrm{Cov}\left[z_{pq},a_j^*\right]
\end{aligned}$$

and the covariances between the a_i^* and the data z_{kl} are

$$\mathrm{Cov}\left[a_i^*,z_{kl}\right] = \mathrm{E}\left[a_i^*z_{kl}\right] - \mathrm{E}\left[a_i^*\right]\mathrm{E}\left[z_{kl}\right] \ .$$

The joint expectations are then computed by substituting the expectation of the reified simulator evaluated at its best input

$$\begin{aligned}
\mathrm{E}\left[a_i^*z_{kl}\right] &= \mathrm{E}\left[a_i^*\mathrm{E}\left[f_k^*\left(a^*,b_l\right)|a^*\right]\right] \\
&= \mathrm{E}_F\left[\beta_{kp}^*\right]\int a_i^*g_p^*\left(a^*,b_l\right)p\left(a^*\right)da^* \\
&\quad + W_{pqr}\int a_i^*\mathrm{Cov}\left[r_k^*\left(a^*,b_l\right),F_{pqr}\right]p\left(a^*\right)da^* \ .
\end{aligned}$$

Feasibility of calculations While the expressions for these quantities are not any more complicated than for the single-level case, evaluating all of the required integrals is potentially more difficult; this is due to the terms involving the covariance function $\mathrm{Cov}\left[r_i^*\left(.\right),r_k^*\left(.\right)\right]$, which is a recursive function composed of weightings $\gamma_{ij}^{(k)}\left(.\right)$ and residual component covariances $\mathrm{Cov}\left[\nu_i^{(k)}\left(.\right),\nu_j^{(k)}\left(.'\right)\right]$ for each level.

This means that integrals involving a single such covariance function actually require the evaluation of n_s times as many integrals as the equivalent integral in Section 2.4.2, and the integral involving a product of two such covariances requires the evaluation of ${n_s}^2$ times as many terms.

In cases where we are dealing with multiple output functions and multiple simulator levels, and we have made a detailed specification for all the required functions, this can represent quite a large additional burden; if we are able to do so, we must algebraically compute and code all of these terms, and if we are not able to do this algebraically, then we must evaluate them all numerically instead.

When designing the reification framework in a particular situation, careful consideration should be given to how we are going to perform these calculations; in this framework, the ability to perform integrals analytically potentially confers large benefits, because there are so many more of them. In cases where we are fitting the model using relatively large numbers of simulator evaluations, then the final terms in both of the above expressions can be particularly expensive to evaluate; we must compute a matrix the size of $\mathrm{Var}\left[F\right]$ for each pair of inputs $\{b, b'\}$ which, for problems involving any significant amount of system data, can be a slow process.

In many cases, therefore, it may be preferable to simply numerically evaluate the required system expectations and covariances directly from the adjusted expectations and covariances for the reified simulator; if this is an expensive operation, then an alternative approach might be to re-emulate these numerically-evaluated terms and then use the emulator predictions for the system directly. The chosen approach should be checked to make sure that it adequately reproduces the properties of the underlying reifying framework.

5.4 Design

We now consider the consequences of a model specified in this way for the design of a sequence of experiments to collect data on the system. If the structure outlined in the preceding sections is used to construct a model for a system, it can introduce additional complications into the sequential design procedure of Section 3.3. There

are two cases to consider:

- in some instances, while we have beliefs about how the current simulator may evolve in the future, we have no plans for development during the time-scale of the proposed sequence of experiments;
- in others, we may have the option (or the commitment) to construct some of the envisaged simulators between particular experimental stages;

In the following sections, we consider the effect on the design procedure in each of these situations.

5.4.1 No evolution

If the proposed additional simulators are not to be developed within the time-frame of the experiments which will collect the system data, then there is no substantial modification which needs to be made to the backward induction procedure; we have simply introduced a more complex forward model structure into the problem.
Within the reifying framework, the system data z_j to be collected at stage j are represented by a combination of the reified simulator $f^*(.)$ and the discrepancy $\delta(.)$, and so beliefs about the system at any given setting of the design and environmental inputs are computed by propagating uncertainty about the best setting a^* of the model parameters through the reified simulator as described in Section 5.3.3; as before, at stage j of the problem, if $q = a^*$ and $b = \{w_j, d_j\}$, then the moments of the z_j (where z_{jk} is the k^{th} element of the j^{th} set of observations) are

$$\begin{aligned} \mathrm{E}[z_{jk}] &= \mathrm{E}[y_k(w_j, d_j)] \\ \mathrm{Cov}[z_{jk}, z_{lp}] &= \mathrm{Cov}[y_k(w_j, d_j), y_p(w_l, d_l)] + \mathrm{Cov}[\epsilon_{jk}, \epsilon_{lp}] \end{aligned}$$

where the system value is a linear combination of the reified simulator evaluated at its best input setting and the discrepancy at this point

$$y_k(w_j, d_j) = \hat{f}_k^*(w_j, d_j) + \delta_k(w_j, d_j)$$

and the expectations and covariances of these system values are as before

$$\begin{aligned} \mathrm{E}\left[y_k\left(w_j, d_j\right)\right] &= \mathrm{E}\left[\hat{f}_k^*\left(w_j, d_j\right)\right] + \mathrm{E}\left[\delta_k\left(w_j, d_j\right)\right] \\ \mathrm{Cov}\left[y_k\left(w_j, d_j\right), y_p\left(w_l, d_l\right)\right] &= \mathrm{Cov}\left[\hat{f}_k^*\left(w_j, d_j\right), \hat{f}_p^*\left(w_l, d_l\right)\right] \\ &\quad + \mathrm{Cov}\left[\delta_k\left(w_j, d_j\right), \delta_p\left(w_l, d_l\right)\right] . \end{aligned}$$

The backward induction algorithm can then be performed as outlined in Section 3.4 (algorithm 2) using the moments $\mathrm{E}\left[z_{jk}\right]$ and $\mathrm{Cov}\left[z_{jk}, z_{lp}\right]$ for the system data; where required, the conditional moments $\mathrm{E}\left[z_j | z_{[j-1]}, w_{[j]}, d_{[j]}\right]$ and $\mathrm{E}\left[w_j | w_{[j-1]}, d_{[j]}\right]$ are approximated using the adjusted moments $\mathrm{E}_{z_{[j-1]}}\left[z_j | w_{[j]}, d_{[j]}\right]$ and $\mathrm{E}_{w_{[j-1]}}\left[w_j | d_{[j]}\right]$ as in Section 3.4.3 (with corresponding approximations for the conditional variances). The main additional complication in this situation, therefore, is the potential for the uncertainty propagation calculations described in Section 5.3.3 to be challenging; otherwise, the procedure is no more difficult than before.

5.4.2 Evolution

If we have the option to build at least one improved simulator at some point during the experimental procedure, then things become more complicated. Under the modelling framework assumed in Section 5.3, when we construct and evaluate an improved simulator, we resolve some of our uncertainty about the system that the simulators represent; our uncertainty will be reduced by different amounts in different parts of the system input space, potentially by amounts which depend on decisions about the development and evaluation of these simulators which we will not make until nearer the time. This resolution of uncertainty at a certain point in the future could affect the designs that we choose for our experiments, or could even change our decision about whether to carry out a particular experiment at all. If we consider the simulator development process as part of our design procedure, then at stage j:

- we consider the simulator development options available to us at stage j, and determine the optimal set of developments;
- we build the improved simulator, and collect a set of runs on it;

- under our current beliefs, we determine the optimal design for the next available experiment, and we consider whether it is worth paying for this experiment, or whether we should just make a decision based on the experimental data and simulator runs collected so far;

- if we consider the next experiment to be worth its price-tag, then we carry it out, before using this information to determine whether it is worth paying for a further round of simulator development, or whether we should just make a decision about the system using the information collected up until now.

In the remainder of this section, we adapt the backward induction procedure presented in Section 3.3 to handle the availability of simulator developments at each stage.

Full backward induction algorithm: In order to adapt the backward induction for the case of evolving simulators, we introduce the following notation (in addition to that introduced in Section 3.3.1):

- We denote the set of simulator developments chosen at stage j by $\phi_j = \{\phi_{j1}, \ldots, \phi_{jn_{\phi_j}}\}$, and we denote the collection of such inputs up to stage j by $\phi_{[j]} = \{\phi_1, \ldots, \phi_j\}$.

- We denote the j^{th} set of simulator runs that we obtain by F_j, with the collection up to stage j denoted by $F_{[j]} = \{F_1, \ldots, F_j\}$; note that we allow for the possibility that we may construct and evaluate multiple simulators at the same stage.

Allowing our simulators to evolve between stages, the analogue of the terminal risk (3.3.6) takes account of the information available from the simulator runs performed up to stage j

$$\rho_j^t\left[z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \min_{a_j \in \mathcal{A}_j} \int L_j\left(q, a_j\right) p\left(q | z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right) dq \quad (5.4.9)$$

where the posterior distribution for the model parameters is

$$p\left(q | z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right) = \frac{p\left(z_{[j]} | q, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right) p\left(q\right)}{p\left(z_{[j]} | w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right)} \quad (5.4.10)$$

where, as in Section 3.3, we have assumed that our prior beliefs about q do not depend on any of $\{w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\}$. The backward induction algorithm then runs as follows, for stages $j = n, (n-1), \ldots, 1$:

- At stage n, the risk is simply the risk from an optimal terminal decision at that stage: $\rho_n \left[z_{[n]}, w_{[n]}, d_{[n]}, F_{[n]}, \phi_{[n]}\right] = \rho_n^t$. Otherwise, we compare the risk from an optimal terminal decision with the risk from future sampling

$$\rho_j \left[z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \min\left[\rho_j^t, \psi_{j+1}^*\right] \tag{5.4.11}$$

  where ψ_{j+1}^* is defined in (5.4.17) at the final step of the algorithm.

- The expectation over observed data and external parameters is then computed in the same way as in (3.3.10)

$$\bar{\rho}_j \left[z_{[j-1]}, w_{[j-1]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \iint \Big[\rho_j p\left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]}, F_{[j]}\right) p\left(w_j | w_{[j-1]}, d_{[j]}\right)\Big] dz_j dw_j \tag{5.4.12}$$

  where, as in Section 3.3 (algorithm 1), we assume that we do not use the data $z_{[j]}$ or the simulator runs $F_{[j]}$ to learn about the external parameters $w_{[j]}$.

- Optimising over d_j then gives the risk from an optimal future procedure, conditional on the simulator runs F_j that we obtained from the new model built and run at this stage

$$\rho_j^* \left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right] = \min_{d_j \in \mathcal{D}_j} \left[\bar{\rho}_j + c_j\left(d_j\right)\right]. \tag{5.4.13}$$

- Upon reaching this stage of the procedure, we will have the option of either making an immediate decision based on the data observed and the simulator runs obtained up until this point, or performing the experiment at stage j and proceeding optimally from that point. As a function of all parameters, this choice has the following risk

$$\psi_j \left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right] = \min\left[\rho_j^*, \rho_j^f\right] \tag{5.4.14}$$

where ρ_j^f is the risk from choosing to make a decision about the system and terminate after constructing and running the simulator $f^{(j)}(.)$

$$\rho_j^f\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right] = \min_{a_j \in \mathcal{A}_j} \int L_j(q, a_j)\, p\left(q | z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right) dq \ . \quad (5.4.15)$$

- This risk is then averaged over our beliefs about the simulator runs F_j conditional on all of the simulators that have so far been constructed and the points at which it has been run

$$\bar{\psi}_j\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j]}\right] = \int \psi_j p\left(F_j | z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j]}\right) dF_j \ . \quad (5.4.16)$$

- Finally, we optimise the risk over the possible simulator development choices ϕ_j, including the cost $c_j^f(\phi_j)$ of choosing to make a particular set of developments

$$\psi_j^*\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j-1]}\right] = \min_{\phi_j}\left[\bar{\psi}_j + c_j^f(\phi_j)\right] . \quad (5.4.17)$$

The procedure is also presented as an algorithm in 4 (and sub-algorithms 5 and 6 describing the parts related to the system experiments and the simulator).

Computational issues To generate an optimal design for the first experiment, taking account of all of the experimental and simulator costs and uncertainties, we must compute all of the functions listed above. Of course, since we could not do this analytically for the simpler problem where the system model was fixed throughout, and where we only needed to consider the experiment (Section 3.3), we certainly cannot do it for this problem. Again, then, we need a procedure for approximating the calculations numerically; we must adapt the procedure presented in Section 3.4 to include the new steps, taking care to ensure that doing so does not render the whole thing too computationally expensive to be carried out.

There are two main additional features of this problem which have the potential to cause computational difficulty; we consider each of these below, and discuss simplification strategies to help us deal with them:

- In general, the simulator development inputs ϕ_j set should contain variables which specify the ways in which we make improvements to the simulator at a particular stage, but also variables which specify the points at which we will run our simulator in order to collect the runs F_j. It is regularly the case that we will have access to many more simulator runs than system data, and so while we may be able to set up the problem so that the development choices have a low-dimensional parametrisation, attempting to design for the simulator runs would make ϕ_j a high-dimensional variable (relative to d_j), and make optimising the risk over this quantity a hard problem. To reduce the dimensionality of this optimisation problem, we simply ignore the simulator run design problem, instead fixing a Latin hypercube design at which we will evaluate the simulator once it is built. This is a common strategy for computer experiments (see, for example, Santner et al. [2002]); instead of attempting to solve the (difficult) design problem posed by a particular simulator, a simple, space-filling design is chosen to give good coverage of the input space.

- In this problem, the data that we observe on the system in our experiments contains not only information about the parameters q, but also information about the simulators that we have developed, or that we might develop in the future. If we were to take account of all this information, then, we would compute the expectation of the risk (equation (5.4.16)) with respect to the posterior distribution $p\left(F_j|z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j]}\right)$ for the simulator runs given all of the information obtained so far. This is a highly complex posterior distribution, and so to simplify the procedure, we do not attempt to evaluate it; instead, in our approximation procedure, we compute the expectation of the risk with respect to the conditional distribution $p\left(F_j|F_{[j-1]}, \phi_{[j]}\right)$ for the current simulator runs given past simulator runs and development choices.

Algorithm 4 Backward induction procedure with simulator development.

for $k = 0, 1, \ldots, (n-1)$ **do**
  for $j = n, (n-1), \ldots, (k+1)$ **do**
    Run algorithm 5: compute the optimal experimental risk

$$\rho_j^* \left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]} \right]$$

    Run algorithm 6: compute the optimal simulator risk

$$\psi_j^* \left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j-1]} \right]$$

    if $\rho_k^t \leq \psi_{k+1}^*$ **then**
      Cease, take immediate decision a_k^* (risk ρ_k^t)
    else
      Pay cost $c_{k+1}^f \left(\phi_{k+1}^* \right)$, make simulator developments ϕ_{k+1}^*, obtain F_{k+1}
      if $\rho_{k+1}^f \leq \rho_{k+1}^*$ **then**
        Cease, take immediate decision a_{k+1}^* (risk ρ_{k+1}^f)
      else
        Pay cost $c_{k+1} \left(d_{k+1}^* \right)$, observe $\{z_{k+1}, w_{k+1}\}$ at d_{k+1}^*
      end if
    end if
  end for
end for

Algorithm 5 Compute the experimental risk, to feed into algorithm 4.

1: Compute the experimental risk

2: **if** $j = n$ **then**

$$\rho_n \left[z_{[n]}, w_{[n]}, d_{[n]}, F_{[n]}, \phi_{[n]}\right] = \rho_n^t \left[z_{[n]}, w_{[n]}, d_{[n]}, F_{[n]}, \phi_{[n]}\right]$$

3: **else**

$$\rho_j \left[z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \min \left[\rho_j^t \left[z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right], \rho_{j+1}^* \left[z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right]\right]$$

4: **end if**

5: Compute the expected experimental risk

$$\bar{\rho}_j \left[z_{[j-1]}, w_{[j-1]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \iint \rho_j p\left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right) \times p\left(w_j | w_{[j-1]}, d_{[j]}\right) dz_j dw_j$$

6: Compute the optimal experimental risk:

$$\rho_j^* \left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right] = \min_{d_j \in \mathcal{D}} \left[\bar{\rho}_j \left[z_{[j-1]}, w_{[j-1]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] + c_j\left(d_j\right)\right]$$

Algorithm 6 Compute the simulator risk, to feed into algorithm 4.

1: Compute the simulator risk

$$\psi_j\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right] = \min\left[\rho_j^*\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right], \rho_j^f\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right]\right]$$

2: Compute the expected simulator risk

$$\bar{\psi}_j\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j]}\right] = \int \psi_j p\left(F_j | z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j]}\right) dF_j$$

3: Compute the optimal simulator risk

$$\psi_j^*\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j-1]}\right] = \min_{\phi_j}\left[\bar{\psi}_j\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j]}\right] + c_j^f\left(\phi_j\right)\right]$$

5.5 Approximate backward induction- evolving models

We now present an approximation procedure for the backward induction calculation presented in Section 5.4, in which we have the option to select simulator modifications as well as experimental designs. The algorithm runs in much the same way as the one presented in Section 3.4 (see algorithm 2); we begin by giving an outline of the approximating procedure in Section 5.5.1, and we provide further detail about each of the required steps in Sections 5.5.2 to 5.5.7.

5.5.1 Outline of the approximating procedure

As in Section 3.4.1, the approximation procedure runs in waves, indexed by $i = 1, 2, \ldots$; at each wave, we re-emulate inside the candidate design space found at the previous wave. The following steps are performed for each stage of the backward induction procedure.

Emulate the experimental risk As before, our first step is to fit an emulator to the risk ρ_j (equation (5.4.11)) from an optimal course of action after the experiment j has been carried out. Denoting our approximation to the risk at stage j at wave i by $r_j^{(i)}[.]$, we assume the common regression, residual and nugget form for the emulator

$$r_j^{(i)}\left[z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \sum_p \alpha_{jp}^{(i)} h_{jp}^{(i)}\left(z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right) + u_j^{(i)}\left(z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right) + \xi_j^{(i)} \quad (5.5.18)$$

where, as in Section 3.4.1, the $h_{jp}^{(i)}(.)$ are a set of known basis functions, and the regression coefficients $\{\alpha_{jp}^{(i)}\}$, residual process $u_j^{(i)}(.)$ and nugget $\xi_j^{(i)}$ are assumed to be a priori independent. We make a prior specification for all of the uncertain components of the model (5.5.18), and we adjust these prior moments using a set of risk evaluations. In Section 5.5.3, we outline a general procedure for making a prior specification, and generating evaluations of the risk for the adjustment. In Section 5.5.7, we consider the selection of the risk inputs that we use to generate data for the adjustment.

Approximate the expected experimental risk The next step of the backward induction algorithm is to compute our beliefs about the expectation of the risk with respect to $p\left(z_j|z_{[j-1]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right)$ and $p\left(w_j|w_{[j-1]}, d_{[j]}\right)$; this is done in exactly the same way as described in Section 3.4.6, by integrating the emulator directly. Our approximation to the expectation $\bar{\rho}_j$ (equation (5.4.12)) of the risk is denoted by $\bar{r}_j^{(i)}$, with

$$\bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \int r_j^{(i)} p\left(z_j|z_{[j-1]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right) p\left(w_j|w_{[j-1]}, d_{[j]}\right) dz_j dw_j \ . \quad (5.5.19)$$

Computation of the moments of $\bar{r}_j^{(i)}$ is discussed in Section 5.5.3, and the characterisation of the distributions required to evaluate the integral is discussed in Section 5.5.2.

Characterise the risk from an optimal experimental design As in Section 3.4.1, we denote our approximation to the risk ρ_j^* (equation (5.4.13)) from an optimal

design by $s_j^{(i)}$, where

$$s_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right] = \bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, \{d_{[j-1]}, d_j^*\}, F_{[j]}, \phi_{[j]}\right] + c_j\left(d_j^*\right) . \tag{5.5.20}$$

As before, we do not know the optimal design setting d_j^*, and so we represent our uncertainty about it by sampling candidate designs $\tilde{d}_j$ from a candidate design space $\mathcal{D}_j^{(i)}$ defined by the emulator $\bar{r}_j^{(i)}$. This corresponds to the strategy adopted in Section 3.4.8, before we allowed for the possibility of evolving models. Its implementation in this instance is discussed in Section 5.5.3.

Emulate the simulator risk We must now model the risk from an optimal choice between performing the next experiment and making a decision about the system immediately after building a new simulator. In order to approximate the simulator risk ψ_j (equation 5.4.14) we re-emulate: we denote our approximating emulator by $t_j^{(i)}$, and we assume the common regression, residual and nugget form for our emulator

$$\begin{aligned} t_j^{(i)}&\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right] \\ &= \sum_p \hat{\alpha}_{jp}^{(i)} \hat{h}_{jp}^{(i)}\left(z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right) \\ &\quad + \hat{u}_j^{(i)}\left(z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right) + \hat{\xi}_j^{(i)} . \end{aligned} \tag{5.5.21}$$

As usual with our emulators, the $\hat{h}_{jp}^{(i)}(.)$ are known basis functions, and the uncertain components (regression coefficients $\{\hat{\alpha}_{jp}^{(i)}\}$, residual process $\hat{u}_j^{(i)}(.)$ and nugget $\hat{\xi}_j^{(i)}$) are assumed to be a priori uncorrelated. We make a prior specification for the uncertain components of the model, and then we adjust these prior beliefs using a set of risk evaluations. The specification of the prior, and the generation of the risk evaluations for the adjustment are discussed in Section 5.5.4.

Approximate the expected simulator risk We denote our approximation to the expected simulator risk $\bar{\psi}_j$ (defined in (5.4.16)) by $\bar{t}_j^{(i)}$; as with the expected experimental risk $\bar{r}_j^{(i)}$, it is computed by integrating the emulator directly

$$\bar{t}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j]}\right] = \int t_j^{(i)} p\left(F_j | F_{[j]}, \phi_{[j]}\right) dF_j . \tag{5.5.22}$$

As with the data and the external parameters, we compute this expectation by integrating the emulator directly, as described in Section 2.4.1. The fitting and integration of the emulator $t_j^{(i)}[.]$ are discussed in Section 5.5.4.

Characterise the risk from an optimal experimental design Our approximation $v_j^{(i)}$ to the risk ψ_j^* (equation (5.4.17)) from an optimally-designed simulator is generated in the same way as our approximation to the risk $s_j^{(i)}$ from an optimally-designed experiment

$$v_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j-1]}\right] = \bar{t}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \{\phi_{[j]}, \phi_j^*\}\right] + c_j^f\left(\phi_j^*\right) . \quad (5.5.23)$$

The optimal simulator design ϕ_j^* is unknown; we represent our uncertainty about its value by sampling candidate simulator designs $\tilde{\phi}_j$ from a candidate simulator design space $\mathcal{P}_j^{(i)}$ defined by the emulator $\bar{t}_j^{(i)}$. Generation of the candidate simulator designs and characterisation of our uncertainty about $v_j^{(i)}$ is discussed in Section 5.5.5.

5.5.2 Characterisation of moments

We need to be able to characterise the posterior distributions $p\left(q|z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right)$ and $p\left(q|z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right)$ of the parameters at any stage of the problem, as well as the conditional distributions $p\left(z_j|z_{[j-1]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right)$ and $p\left(F_j|F_{[j-1]}, \phi_{[j]}\right)$. If we were to use a fully probabilistic specification for all of the components of the model, then we would need to resort to computationally expensive numerical integration methods in order to characterise these distributions. The fact that we can represent our model as a DAG would help slightly in this, as it would allow us to construct an MCMC scheme in which we can sample from each of the conditional distributions in turn; however, this scheme would still suffer from all of the usual issues (difficulty in identifying convergence, long computation times).

Instead, we adopt the same strategy as in Section 3.4.3 and use the second-order moments provided by our Bayes linear analysis to characterise the moments of the required distributions at each stage.

Algorithm 7 Backward induction approximation algorithm, with evolving models.

1: **for** $i = 1, 2, \ldots$ **do** (loop over waves)

2: **for** $j = n, (n-1), \ldots, 1$ **do** (loop over stages)

3: Specify experimental risk model (Section 5.5.3)

$$r_j^{(i)}\left[z_{[j]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \sum_p \alpha_{jp}^{(i)} h_{jp}^{(i)} + u_j^{(i)} + \xi_j^{(i)}$$

4: Approximate expected experimental risk (Section 5.5.3)

$$\bar{r}_j^{(i)} = \iint r_j^{(i)} p\left(z_j | z_{[j-1]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right) p\left(w_j | w_{[j-1]}, d_{[j]}\right) dz_j dw_j$$

5: Characterise the risk from an optimal experiment (Section 5.5.3)

$$s_j^{(i)} = \bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, \{d_{[j-1]}, d_j^*\}, F_{[j]}, \phi_{[j]}\right] + c_j\left(d_j^*\right)$$

6: Specify simulator risk model (Section 5.5.4)

$$t_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right] = \sum_p \hat{\alpha}_{jp}^{(i)} \hat{h}_{jp}^{(i)} + \hat{u}_j^{(i)} + \hat{\xi}_j^{(i)}$$

7: Approximate expected simulator risk (Section 5.5.4)

$$\bar{t}_j^{(i)} = \int t_j^{(i)} p\left(F_j | F_{[j]}, \phi_{[j]}\right) dF_j$$

8: Characterise the risk from an optimally-designed simulator (Section 5.5.5)

$$v_j^{(i)} = \bar{t}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \{\phi_{[j]}, \phi_j^*\}\right] + c_j^f\left(\phi_j^*\right)$$

9: **end for**

10: **end for**

Marginal data moments First, the observed data; the marginal moments of the data z_j available at the j^{th} stage are computed for fixed w_j, d_j and $F_{[j]}$

$$\begin{aligned}\mathrm{E}\left[z_{jk}|w_j, d_j, F_{[j]}\right] &= \mathrm{E}\left[y_k\left(w_j, d_j\right)\right] \\ \mathrm{Cov}\left[z_{jk}, z_{lp}|w_j, d_j, F_{[j]}\right] &= \mathrm{Cov}\left[y_k\left(w_j, d_j\right), y_p\left(w_l, d_l\right)\right] \\ &\quad + \mathrm{Cov}\left[\epsilon_{jk}, \epsilon_{lp}\right]\end{aligned}$$

where the moments of $y\left(.\right)$ are computed as in Section 5.3.2. The adjusted moments $\mathrm{E}_{z_{[j-1]}}\left[z_j|w_{[j]}, d_{[j]}, F_{[j]}\right]$ and $\mathrm{Var}_{z_{[j-1]}}\left[z_j|w_{[j]}, d_{[j]}, F_{[j]}\right]$ are computed from these marginal moments as

$$\mathrm{E}_{z_{[j-1]}}\left[z_{jk}|, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \mathrm{E}\left[z_{jk}\right] + \mathrm{Cov}\left[z_{jk}, z_{rs}\right]\mathrm{Var}\left[z_{[j-1]}\right]^{-1}_{rstu}\Big[z_{tu} - \mathrm{E}\left[z_{tu}\right]\Big]$$

$$\mathrm{Cov}_{z_{[j-1]}}\left[z_{jk}, z_{jp}|w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \mathrm{Cov}\left[z_{jk}, z_{jp}\right] - \mathrm{Cov}\left[z_{jk}, z_{jp}\right]\mathrm{Var}\left[z_{[j-1]}\right]^{-1}_{rstu}\mathrm{Cov}\left[z_{tu}, z_{jp}\right]$$

where the sums over r and t range from $1, \ldots, (j-1)$, and the sums over s and u range over $1, \ldots, n_{z_s}$ and $1, \ldots, n_{z_u}$ respectively, and all of the moments on the right hand side are conditioned on $\{w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\}$. These adjusted moments are used to approximate the conditional distributions $p\left(z_j|z_{[j-1]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right)$; we simply use these adjusted moments to characterise a multivariate Gaussian distribution.

Posterior parameter moments We characterise the posterior distribution at each stage by computing the adjusted moments of the model parameters given the data, as in Section 5.3.3

$$\mathrm{E}_{z_{[j]}}\left[q_l|w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \mathrm{E}\left[q_l\right] + \mathrm{Cov}\left[q_l, z_{rs}\right]\mathrm{Var}\left[z_{[j]}\right]^{-1}_{rstu}\Big[z_{tu} - \mathrm{E}\left[z_{tu}\right]\Big]$$

$$\mathrm{Cov}_{z_{[j]}}\left[q_l, q_m|w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right] = \mathrm{Cov}\left[q_l, q_m\right] - \mathrm{Cov}\left[q_l, z_{rs}\right]\mathrm{Var}\left[z_{[j]}\right]^{-1}_{rstu}\mathrm{Cov}\left[z_{tu}, q_m\right]$$

where the sums over r and t range from $1, \ldots, j$, the sums over s and u range over $1, \ldots, n_{z_r}$ and $1, \ldots, n_{z_t}$ respectively, and all moments on the right-hand side are

conditioned on $\{w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\}$. These moments are then used to characterise an appropriate distribution, usually a multivariate Gaussian. In addition to these adjusted moments computed after specification of $\{z_{[j]}, w_{[j]}, w_{[j]}, F_{[j]}, \phi_{[j]}\}$ (after the experiment at stage j), we must also be able to approximate the corresponding adjusted moments after specification of $\{z_{[j-1]}, w_{[j-1]}, w_{[j-1]}, F_{[j]}, \phi_{[j]}\}$, in order to characterise the posterior after the simulator developments at stage j, but before the j^{th} experiment has been carried out. These are approximated using the adjusted moments computed in exactly the same way, simply adjusting the conditioning sets and the ranges of the summation indices appropriately.

Conditional simulator moments We must also be able to specify the distributions $p\left(F_j|F_{[j-1]}, \phi_{[j]}\right)$ of the simulator runs that we obtain at stage j; we specify that each of these is a multivariate Gaussian distribution, characterised by the adjusted second order moments for the simulator runs computed as at 5.3.2. The simulator that we choose to construct at stage j is determined by the parameters ϕ_j; within this, we allow for the possibility that we may choose to construct multiple simulators at a given stage, or that we may choose to construct none at all. In any case, the joint moments of any runs that we do obtain can be computed as at 5.3.2.

5.5.3 Emulating the experimental risk

In this section, we discuss the fitting of the emulator $r_j^{(i)}$ to the experimental risk ρ_j^t

Evaluating the risk At the j^{th} stage of the i^{th} wave of the approximation procedure, we evaluate the risk at $N_j^{(i)}$ input settings $\{z_{[j]k}, w_{[j]k}, d_{[j]k}, F_{[j]k}, \phi_{[j]k}\}$ to obtain risk evaluations $R_j^{(i)} = \{R_{1k}^{(i)}, \ldots, R_{jN_j^{(i)}}^{(i)}\}$. At the final stage, these risk evaluations are obtained directly from the terminal risk (5.4.9)

$$R_{nk}^{(i)} = \rho_n^t \left[z_{[n]k}, w_{[n]k}, d_{[n]k}, F_{[n]k}, \phi_{[n]k}\right] \ .$$

At any other stage of the algorithm, the evaluations are obtained by comparing the terminal risk with our approximation (5.5.23) to the risk from an optimal future

procedure

$$R_{jk}^{(i)} = \min \Big[\rho_j^t \left[z_{[j]k}, w_{[j]k}, d_{[j]k}, F_{[j]k}, \phi_{[j]k} \right], \\ v_{j+1}^{(i)} \left[z_{[j]k}, w_{[j]k}, d_{[j]k}, F_{[j]k}, \phi_{[j]k} \right] \Big] . \quad (5.5.24)$$

The uncertainty induced through our numerical approximations means that $v_{j+1}^{(i)}$ is unknown, which means that we cannot evaluate $R_{jk}^{(i)}$ exactly; therefore, as in Section 3.4.4, we compute $\mathrm{E}\left[R_{jk}^{(i)}\right]$ and $\mathrm{Cov}\left[R_{jk}^{(i)}, R_{jl}^{(i)}\right]$ by using the moments $\mathrm{E}\left[v_{j+1}^{(i)}[.]\right]$ and $\mathrm{Cov}\left[v_{j+1}^{(i)}[.], v_{j+1}^{(i)}[.']\right]$ computed in Section 5.5.5 to characterise a multivariate Gaussian distribution and sampling the expression (5.5.24). The emulator is then fitted to the expected values, with the covariance structure being used to characterise the measurement error.

Model specification The procedure for specifying and fitting the emulator for the experimental risk is the same as the one outlined in Section 3.4.4. First, we select the basis functions $\{h_{jp}^{(i)}\}$ and specify the covariance function $\mathrm{Cov}\left[u_j^{(i)}(.), u_j^{(i)}(.')\right]$ for the residual component. Then, we make a prior specification for the basis coefficients $\{\alpha_{jp}^{(i)}\}$ and the marginal variance of the residual process by performing an initial linear regression. The correlation parameters of the residual covariance function are determined through leave-one-out cross-validation; see Sections 2.2.3 and 3.4.4 for further details of this process.

As previously discussed (Section 3.4.4), using a modified version of the risk as a basis function may help to explain a substantial proportion of the global structure of the risk function. However, in this case, the complexity of the general model structure outlined in Section 5.3 has the potential to make the evaluating risks (5.4.9) or (5.4.15) too computationally intensive for them to be practical as basis function choices; when sampling candidate designs (below), or candidate simulator designs (Section 5.5.5), we will generally have to evaluate the emulator many thousands of times at different input settings.

Instead of using the risks (5.4.9) or (5.4.15) directly as a component of the basis, then, we might opt to use a simplified version of the risk. For example, we could:

- simplify the uncertainty propagation calculations in Section 5.3.3 by dropping

terms; for example, the integrals involving the covariance twice can be particularly slow to evaluate, since we must compute a large matrix for each pair of observations;

- reduce the number of points used when numerically evaluating the moments in Section 5.3.3;
- simplify the risk by using only a sub-sample of the data $z_{[j]}$ to evaluate it;
- use the risk from a simplified problem (for example, one in which we do not consider improvements to the model) which shares many of the features of the actual problem.

Approximating the expected experimental risk Once the emulator $r_j^{(i)}$ has been fitted, the moments of our approximation to the expected risk are characterised exactly as outlined in Section 3.4.6, by integrating with respect to the distributions $p\left(z_j|z_{[j-1]}, w_{[j]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right)$ and $p\left(w_j|w_{[j-1]}, d_{[j]}\right)$; in this case, we must allow for the dependence of our emulator and data distribution on $\{F_{[j]}, \phi_{[j]}\}$, the simulator runs and simulator design choices.

Characterise the risk from an optimal experimental design Once we have computed the moments of $\bar{r}_j^{(i)}$ as outlined in Section 3.4.6, our procedure for generating candidate designs $\tilde{d}_j$ and assessing the moments of $s_j^{(i)}$ is also the same as outlined in Section 3.4.8; again, in this case, we must take account of any dependency of our fitted model on the simulator components $\{F_{[j]}, \phi_{[j]}\}$.

5.5.4 Emulating the simulator risk

In the procedure where we may build simulators in order to resolve parts of the uncertainty about the system, we must account for the effect of the simulator runs that we might observe at this stage on our beliefs about the system. In order to do this, we must characterise the risk ψ_j (defined in equation (5.4.14)), and its expected behaviour $\bar{\psi}_j$ over possible values of F_j. Since, again, the risk ψ_j is computed as the minimum of two already very complex functions, we emulate again in order to

proceed; the emulator that we fit has the common regression, residual and nugget form (see equation (5.5.21)).

Evaluating the risk In order to fit the model, we need to generate data from the risk function. We evaluate the risk at a set of $\tilde{N}_j^{(i)}$ input settings $\{z_{[j]k}, w_{[j]k}, d_{[j]k}, F_{[j]k}, \phi_{[j]k}\}$, and we denote the risk values that we obtain at these points by $\{T_{jk}^{(i)}\}$. The risk evaluations are generated by comparing the risk ρ_j^f (equation (5.4.15)) from a decision immediately after the construction of the j^{th} set of simulators with our approximation $s_j^{(i)}$ (equation (5.4.13)) to the risk from an optimally-designed experiment at stage j

$$T_{jk}^{(i)} = \min\left[\rho_j^f\left[z_{[j]k}, w_{[j]k}, d_{[j]k}, F_{[j]k}, \phi_{[j]k}\right], s_j^{(i)}\left[z_{[j]k}, w_{[j]k}, d_{[j]k}, F_{[j]k}, \phi_{[j]k}\right]\right]. \tag{5.5.25}$$

Since we are uncertain about $s_j^{(i)}$, we compute $\mathrm{E}\left[T_{jk}^{(i)}\right]$ and $\mathrm{Cov}\left[T_{jk}^{(i)}, T_{jl}^{(i)}\right]$ by sampling. We compute $\rho_j^f[.]$ for any setting of its inputs by evaluating (5.4.15), and we compute $\mathrm{E}\left[s_j^{(i)}[.]\right]$ and $\mathrm{Cov}\left[s_j^{(i)}[.], s_j^{(i)}[.']\right]$ for any pair of input settings as outlined in Section 5.5.3; we use these moments to characterise a multivariate Gaussian distribution ove all input settings, and then assess the moments of $T_{jk}^{(i)}$ by repeatedly sampling the expression (5.5.25).

Choosing the basis functions All previous considerations (Sections 5.5.3 and 3.4.4) apply in this instance too; however, this time, we are looking to ensure that we can handle integrals of our basis and covariance functions with respect to the distribution $p\left(F_j|F_{[j-1]}, \phi_{[j]}\right)$ of simulator runs at this stage.
If it is computationally feasible to do so, then choosing

$$\hat{h}_j^{(i)}\left(z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right) = \rho_j^f\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, \tilde{F}_{[j]}, \phi_{[j]}\right]$$

where $\tilde{F}_{[j]} = \{F_1, \ldots, \mathrm{E}_{F_{[j-1]}}\left[F_j|\phi_{[j]}\right]\}$ is the set of simulator runs with F_j replaced with its adjusted expectation, may provide a good approximation to the risk. The variability in the risk due to the actual values of the simulator runs can then be absorbed using either the residual process $\hat{u}_j^{(i)}$ or the nugget term $\hat{\xi}_j^{(i)}$.
Evaluating the above basis function requires us to compute the adjusted moments of

F_j given $F_{[j-1]}$, and then to re-compute the adjusted moments of the reified simulator for each value of $\tilde{F}_{[j]}$. This could potentially represent too large a computational burden (given the need to evaluate this emulator many times in order to carry out the minimum sampling procedure described in Section 5.5.5); instead then, we could simply choose to use the terminal risk ρ^f_{j-1} evaluated at the expectation of the simulator runs from all stages

$$\hat{h}^{(i)}_j\left(z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\right) = \rho^f_j\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, \mathrm{E}\left[F_{[j]}|\phi_{[j]}\right], \phi_{[j]}\right] .$$

This means that, in order to repeatedly evaluate the risk (5.4.15), we do not need to keep re-updating the moments of the reified simulator as the set of simulator runs changes.

Under either of these basis function choices, we sacrifice some of our ability to explain the variation in the risk function in order to achieve integrability of the resulting emulator and to reduce the time and effort that it takes to evaluate the approximation. For a specific problem, the effect of each of these choices of basis function should be carefully assessed, so that we may make a judgement about the appropriate trade-off between computational effort and uncertainty about the risk.

Determining the prior Once the basis functions have been selected, the prior is determined in the usual way (see Section 3.4.4); a set of risk evaluations is generated, and we use this to perform an initial linear regression. The parameter estimates from this fit are then used to fix the prior moments $\mathrm{E}\left[\hat{\alpha}^{(i)}_{jp}\right]$ and $\mathrm{Cov}\left[\hat{\alpha}^{(i)}_{jp}, \hat{\alpha}^{(i)}_{jq}\right]$, and the residuals of the regression are then used to determine the marginal variance of the residual process $\mathrm{Var}\left[\hat{u}^{(i)}_j(.)\right]$.

If a nugget is used, its variance $\mathrm{Var}\left[\hat{\xi}^{(i)}_j\right]$ is determined empirically as in the original design procedure (Section 3.4.4); we generate risk evaluations where only the parameters whose effect is to be represented by the nugget are allowed to vary, and we compute the sample variances of these risk data sets at multiple different settings of the fixed parameters. If the variances do not differ greatly across the locations, then we fix $\mathrm{Var}\left[\xi^{(i)}_j\right]$ to the average of the sample variances.

Once the prior for the regression, the marginal variance of the residual and the variance of the nugget have been fixed as above, we determine the correlation lengths for

the residual process using leave-one-out cross-validation (see Sections 2.2.3, 3.4.4). For a particular setting of the correlation parameters, we omit each individual element of the data set $T_j^{(i)}$ in turn, and predict its value using the fit to the remainder; we compare the quality of different correlation parameter settings using the sum of the predictive Gaussian likelihoods for all elements. We then fix the correlation lengths to the best-performing setting for the remainder of the analysis.

Approximate the expected simulator risk Once we have fitted the emulator $t_j^{(i)}$, we integrate it to obtain $\bar{t}_j^{(i)}$, our approximation to the risk $\bar{\psi}_j$ at the i^{th} wave of the algorithm. As discussed in Section 2.4.2, and implemented for the experimental risk in Section 3.4.6, moments of the integral of a function are obtained by integrating the moments directly; we have that

$$\mathrm{E}_{T_j^{(i)}}\left[\bar{t}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j]}\right]\right] = \int \mathrm{E}_{T_j^{(i)}}\left[t_j^{(i)}\left[.\right]\right] p\left(F_j | F_{[j-1]}, \phi_{[j]}\right) dF_j$$

and that

$$\mathrm{Cov}_{T_j^{(i)}}\left[\bar{t}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j]}\right], \bar{t}_j^{(i)}\left[z'_{[j-1]}, w'_{[j-1]}, d'_{[j-1]}, F'_{[j-1]}, \phi'_{[j]}\right]\right] = \int \mathrm{Cov}_{T_j^{(i)}}\left[t_j^{(i)}\left[.\right], t_j^{(i)}\left[.'\right]\right] p\left(F_j | F_{[j-1]}, \phi_{[j]}\right) p\left(F'_j | F'_{[j-1]}, \phi'_{[j]}\right) dF_j dF'_j \ .$$

In order to compute the integrals of these expectations and covariances, we must compute the integrals of the basis and and covariance functions, as outlined in Section 2.4.2. If we took care to design our emulator so that these functions could be integrated analytically with respect to $p\left(F_j | F_{[j-1]}, \phi_{[j]}\right)$, then evaluating the moments of $\bar{t}_j^{(i)}$ will be no more complex than evaluating those of $t_j^{(i)}$. In general, we will approximate the distribution $p\left(F_j | F_{[j-1]}, \phi_{[j]}\right)$ by using the adjusted moments $\mathrm{E}_{F_{[j-1]}}\left[F_j | \phi_{[j]}\right]$ and $\mathrm{Var}_{F_{[j-1]}}\left[F_j | \phi_{[j]}\right]$ (computed as in Section 5.3.2) to characterise a multivariate Gaussian.

5.5.5 Characterising the optimal simulator risk

Because of our uncertainties about the risk functions, the optimal set of simulator developments ϕ_j^* cannot be located exactly; for the same reasons as outlined in

Section 3.4.8, it is also not possible to obtain an exact characterisation of our uncertainty about the risk at the minimum. In order to assess our uncertainty about $v_j^{(i)}$, then, we adopt a procedure analogous to the one used when characterising our uncertainty about $s_j^{(i)}$; we use a sampling procedure to identify simulator designs which could plausibly be optimal, and we use designs generated in this way to assess our uncertainty about $v_j^{(i)}$.

Candidate simulator design space We represent our uncertainty about the optimal simulator design ϕ_j^* by using the emulator $\bar{t}_j^{(i)}$ to define a 'candidate simulator design space' $\mathcal{P}_j^{(i)}$. For any setting of the risk inputs $\{z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j-1]}\}$, we generate a space-filling set of $\tilde{M}_j^{(i)}$ trial simulator designs $\{\phi_{j1}, \phi_{j2}, \ldots, \phi_{j\tilde{M}_j^{(i)}}\}$ inside the space $\mathcal{P}_j^{(i-1)}$ (where $\mathcal{P}_j^{(0)} = \mathcal{P}_j$, the full simulator design space), and we sample corresponding risks $\bar{t}_j^{(i)}$ from a Multivariate Gaussian characterised by our beliefs $\mathrm{E}\left[\bar{t}_j^{(i)}\right]$. Our 'candidate simulator design' is the trial simulator design which minimises the risk over this set. The space $\mathcal{P}_j^{(i)}$ is the set of simulator designs which are identified by this procedure. This procedure is summarised in algorithm 8.
As with the procedure for sampling candidate experimental designs (Section 3.4.8, algorithm 3), as defined, this procedure is recursive; we must be able to generate simulator designs which lie in the space $\mathcal{P}_j^{(i-1)}$ before we can generate designs which lie in $\mathcal{P}_j^{(i)}$. As we progress through the waves, this will cause the complexity of the procedure to increase substantially. Generally, however, the candidate simulator designs which we obtain will lie within easily-identifiable sub-regions of the full design space, which we can characterise using simple limits; therefore, we can approximate the full procedure by characterising $\mathcal{P}_j^{(i-1)}$ in this way, and then generating points within this space using a Latin hypercube.

Moments of $v_j^{(i)}$ As with the optimal experimental risk (Section 3.4.8), we use the sampling procedure to characterise our uncertainty about $v_j^{(i)}$, our approximation to ψ_j^* at wave i; we simply substitute candidate designs $\tilde{\phi}_j$ generated according to algorithm 8 for the true optimal simulator design ϕ_j^* in the expression (5.4.17), and

Algorithm 8 Generate candidate simulator design $\tilde{\phi}_j$ at stage j wave i.

1: Generate $\tilde{M}_j^{(i)}$ space-filling trial simulator designs $\{\phi_{j1}, \phi_{j2}, \ldots, \phi_{j\tilde{M}_j^{(i)}}\}$ within the candidate design space $\mathcal{P}_j^{(i-1)}$

2: Jointly sample $\bar{t}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \{\phi_{[j-1]}, \phi_{jk}\}\right]$ values for the set of all trial simulator designs $\{\phi_{jk}\}$ from a Gaussian distribution.

3: Set

$$\tilde{\phi}_j = \arg\min_{\phi_{jk}} \left[\bar{t}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \{\phi_{[j-1]}, \phi_{jk}\}\right] + c_j^f\left(\phi_{jk}\right)\right]$$

compute expectations and covariances by sampling. Our expectation is

$$\mathrm{E}\left[v_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j-1]}\right]\right] = \mathrm{E}\left[\mathrm{E}_{T_j^{(i)}}\left[\bar{t}_j^{(i)}\left[\tilde{\phi}_j\right]\right] + c_j^f\left(\tilde{\phi}_j\right)\right] \tag{5.5.26}$$

and our covariance is

$$\begin{aligned}\mathrm{Cov}\Big[v_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j-1]}\right], v_j^{(i)}\left[z'_{[j-1]}, w'_{[j-1]}, d'_{[j-1]}, F'_{[j-1]}, \phi'_{[j-1]}\right]\Big] \\ = \mathrm{Cov}\left[\mathrm{E}_{T_j^{(i)}}\left[\bar{t}_j^{(i)}\left[\tilde{\phi}_j\right]\right] + c_j^f\left(\tilde{\phi}_j\right), \mathrm{E}_{T_j^{(i)}}\left[\bar{t}_j^{(i)}\left[\tilde{\phi}_j\right]\right] + c_j^f\left(\tilde{\phi}_j\right)\right] \\ + \mathrm{E}\left[\mathrm{Cov}_{T_j^{(i)}}\left[\bar{t}_j^{(i)}\left[\tilde{\phi}_j\right], \bar{t}_j^{(i)}\left[\tilde{\phi}_j\right]\right]\right]\end{aligned} \tag{5.5.27}$$

where all outer expectations and covariances are computed with respect to $\tilde{\phi}_j$

5.5.6 Assessing the result

After performing a wave of the analysis described in this section, we must use the results to decide how to proceed in the actual design problem. This time, however, we have two different situations in which we must carry out an assessment. In the first case, the decision is the same as before (Section 3.4.10); should we carry out an experiment at stage j, or should we make an immediate decision based on our current beliefs? In the other case, we must determine whether, having carried out an experiment at stage j, we should construct the simulator at stage $(j+1)$, or whether we should make a terminal decision using the information from the simulators already constructed and the experiments already performed. We consider both of these cases below.

Assessing the value of constructing a simulator At the point where we have obtained the $(k-1)^{\text{th}}$ set of simulator runs, and carried out the $(k-1)^{\text{th}}$ experiment, we must next decide whether we should carry out further simulator developments, or whether we should stop and make an immediate decision based on our current beliefs. In order to make this choice, we must first decide what we would do if we were to decide to continue development, so that we can compare the risk from this course of action with the risk from stopping.

Since we are uncertain about the risk function, the best that we can do is to find the minimum of our expectation for the risk, taking into account the cost of a particular set of developments

$$\hat{\phi}_k = \arg\min_{\phi_k} \left[\mathrm{E}_{T_k^{(i)}} \left[\bar{t}_k^{(i)} \left[\phi_k \right] \right] + c_k^f \left(\phi_k \right) \right] .$$

Having determined that we would carry out the developments $\hat{\phi}_k$ if we were to proceed, we compare the risk from stopping with the expected risk from continuing: if

$$\mathrm{E}_{T_k^{(i)}} \left[\bar{t}_k^{(i)} \left[\hat{\phi}_k \right] \right] + c_k^f \left(\hat{\phi}_k \right) < \rho_{k-1}^t$$

then we conclude that it is optimal to pay for the next set of simulator developments (at setting $\hat{\phi}_k$), and then to proceed optimally from this point; if the converse is true, then we conclude that it is optimal to make a decision about the system based on our current beliefs, and that it is not worth continuing beyond this point.

As with the original sequential design procedure (Section 3.4.10), we compute the expected value of perfect information (EVPI) for the risk calculation, in order to help us to decide whether a further wave of analysis would help. If we knew the risk function $v_k^{(i)}$ and the optimal simulator design ϕ_k^* exactly, then our risk from an optimal course of action would be

$$\min \left[\rho_{k-1}^t, v_k^{(i)} \left[\phi_k^* \right] + c_k^f \left(\phi_k^* \right) \right] .$$

Suppose that after deciding to carry out simulator developments at setting $\hat{\phi}_k$, we learn ϕ_k^*, the true optimal simulator design setting; our expectation for the loss that

we incur by choosing to experiment at $\hat{\phi}_k$ rather than at ϕ_k^* is

$$\tilde{v}_k^{(i)} = \min\left[\rho_{k-1}^t, \mathrm{E}_{T_k^{(i)}}\left[\bar{t}_k^{(i)}\left[\hat{\phi}_k\right]\right] + c_k^f\left(\hat{\phi}_k\right)\right] - \mathrm{E}\left[\min\left[\rho_{k-1}^t, v_k^{(i)}\left[\phi_k^*\right] + c_k^f\left(\phi_k^*\right)\right]\right] .$$

As in Section 3.4.10, we approximate the expectation of the second term by sampling candidate designs $\tilde{\phi}_k$.

As before (Section 3.4.10), this quantity should be compared to the cost of running another wave of the risk analysis, and the likely benefits from doing so. The EVPI $\tilde{v}_k^{(i)}$ constitutes a upper bound on the amount that we should be willing to pay for further computation, and so we must make a judgement about the likely benefit that we will obtain from another wave; if c_{wv} is our assessment of the cost of further computation, and we expect that the EVPI after this additional wave will be $\mathrm{E}\left[\tilde{v}_k^{(i+1)}\right]$, then we should pay for the additional wave if $c_{\mathrm{wv}} < \tilde{v}_k^{(i)} - \mathrm{E}\left[\tilde{v}_k^{(i+1)}\right]$, and proceed with the decision problem otherwise.

Assessing the value of the next experiment If we have chosen to develop the simulator according to ϕ_k, and we have observed F_k, then we must decide whether it is now optimal to continue and carry out the k^{th} experiment, or whether we should stop and make a decision about the system. The procedure at this point is exactly the same as the one described in Section 3.4.10.

First, we find the risk from the experiment that we would actually choose; again, we don't know the risk function, and so we choose the experimental design which has the lowest expected risk. We denote by $\hat{d}_k$ the design setting which minimises $\mathrm{E}\left[\bar{r}_k^{(i)}\left[d_k\right]\right] + c_k\left(d_k\right)$; at this design setting, we make an immediate decision based on the information available from the simulators run experiments carried out so far if $\mathrm{E}\left[\bar{r}_k^{(i)}\left[\hat{d}_k\right]\right] + c_k\left(\hat{d}_k\right) \geq \rho_k^f$, and we perform the k^{th} experiment if the converse is true.

In the same way as for the original design procedure, we can compute the expected value of perfect information $v_k^{(i)}\left(\hat{d}_k\right)$ about the risk, to help us make a decision about whether we should pay for another wave of analysis; see Section 3.4.10 for details of the calculation.

5.5.7 Choosing inputs for risk evaluations

As with the approximate backward induction framework presented in Section 3.4, the fitting of multiple different emulators to the same risk functions across different waves provides us with the opportunity to select the risk evaluations used for fitting so that the emulator will be as informative as possible for our analysis at a given wave. In the sequential design problem, there may be parts of the design space that we will never reach, because the reduction in the risk from an experiment at such a design is almost certainly less than the cost of carrying out an experiment there; when building our emulator for the risk, then, we want to avoid fitting it in these parts of the space, since this information will never be useful for our analysis.
When considering the sequential design problem with the potential for model development, we must select both experimental design parameters d_j and ϕ_j at a particular stage j in order to be able to evaluate the risk; we consider the choice of suitable settings for both types of design parameter.

Selecting d_j At wave $i = 1$, we simply generate design inputs d_j according to a Latin hypercube. For later waves $i = 2, 3, \ldots$, we generate design choices that fulfil both of the following criteria:

- They could be candidate designs $\tilde{d}_j$ (sampled according to Section 3.4.8, algorithm 3) for the current input setting $\{z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\}$.
- For later stages $j > 1$, they satisfy the following

$$\frac{\mathrm{E}\left[\bar{r}_j^{(i-1)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right]\right] + c_j\left(d_j\right) - \rho_{j-1}^{f}}{\left[\mathrm{Var}\left[\bar{r}_j^{(i-1)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}, F_{[j]}, \phi_{[j]}\right]\right]\right]^{1/2}} \leq 3 \ .$$

  That is, under our beliefs about $\bar{r}_j^{(i-1)}[.]$ at wave $i-1$, there is a greater than 5% chance (again using the three-sigma rule of Pukelsheim [1994]) that we will continue to experiment at design d_j, at stage j, given setting $\{z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j]}, \phi_{[j]}\}$ for all inputs at previous stages.

Selecting ϕ_j At wave $i = 1$, we also generate simulator design inputs ϕ_j according to a Latin hypercube. At later waves $i = 2, 3, \ldots$, we generate simulator design

settings which fulfil both of the following criteria

- They could be candidate simulator designs $\tilde{\phi}_j$ (sampled according to Section 3.4.8, algorithm 3) for the current input setting $\{z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j-1]}\}$.
- For later stages $j > 1$, they satisfy the following

$$\frac{\mathrm{E}\left[\bar{t}_j^{(i-1)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j]}\right]\right] + c_j^f\left(\phi_j\right) - \rho_{j-1}^t}{\left[\mathrm{Var}\left[\bar{t}_j^{(i-1)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}, F_{[j-1]}, \phi_{[j]}\right]\right]\right]^{1/2}} \leq 3 \ .$$

5.6 Example: Building a model

We illustrate the procedure from Section 5.5 through application to an example related to the atmospheric dispersion model outlined in Section 1.1. In Section 5.6.1, we develop a stochastic representation of the transport of particulate matter through the atmosphere, which allows for the possibility of deviations from the standard model outlined in Section 1.1. Then, in Section 5.6.2, we combine this model with the framework outlined in Section 5.3 to link our current version of the plume to our beliefs about developments to the model that we might make in the future. In Section 5.7, we then run the backward induction procedure from Section 5.5 for this model, and discuss the results.

5.6.1 Stochastic atmospheric dispersion model

We consider what happens to a particle of gas which is released from a source in a particular location; using the orthonormal co-ordinates $\omega = (\omega_x, \omega_y, \omega_z)^{\mathrm{T}}$ (which are aligned with the direction of the wind, as defined in Section 1.1), we assume that the particle moves along the direction of the wind at a constant rate, while perpendicular to the wind (in the horizontal plane), it undergoes fractional Brownian motion (see, for example, Yin [1996]); i.e. it follows a Gaussian process $\chi_y(\omega_x)$ which has $\mathrm{E}\left[\chi_y(\omega_x)\right] = 0$ for all ω_x and

$$\mathrm{Cov}\left[\chi_y(\omega_x), \chi_y(\omega_x')\right] = c_y(\omega_x, \omega_x') = \frac{\sigma_y^2}{2}\left[(\omega_x)^{2H_y} + (\omega_x')^{2H_y} - |\omega_y - \omega_x'|^{2H_y}\right]$$

where between them, the marginal standard deviation σ_y and the Hurst parameter H_y govern the distribution of independent increments of the process (the increment

$\chi_y(\omega_x + t) - \chi_y(\omega_x)$ has variance $t^{(2H_y)}\sigma_y^2$). We now assume that N independent such particles are released, and we use Gaussian kernels to construct a measure of the density of particles at a particular location ω_y for a given point ω_x on the trajectory

$$\alpha(\omega_x, \omega_y) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{\sqrt{2\pi}\eta_y} \exp\Big[-\frac{1}{2\eta_y^2}(\omega_y - \chi_{yi}(\omega_x))^2 \Big]$$

where η_y parametrises the roughness of the cross-section. We imagine that, as N grows, the average will converge to a Gaussian; indeed, if we compute the expectation of this smoother over the distribution of the driving process for given $\chi_y(\omega_x)$, we obtain

$$\begin{aligned}
\mathrm{E}[\alpha(\omega_x, \omega_y)] &= \frac{1}{N}\sum_i \int_{-\infty}^{\infty} \mathcal{N}\left(\omega_y | \chi_{yi}(\omega_x), \eta_y^2\right) \mathcal{N}(\chi_{yi}(\omega_x) | 0, v_y(\omega_x)) \, d\chi_y(\omega_x) \\
&= \frac{1}{\sqrt{2\pi(\eta_y^2 + v_y(\omega_x))}} \exp\Big[-\frac{\omega_y^2}{2(\eta_y^2 + v_y(\omega_x))} \Big]
\end{aligned}$$

where the marginal variance of the process at a given point is denoted by

$$v_y(\omega_x) = \frac{\sigma_y^2}{2}\omega_x^{(2H_y)} .$$

The Gaussian form also allows us to compute the covariance between the particle densities at different points

$$\begin{aligned}
\mathrm{Cov}\left[\alpha(\omega_x, \omega_y), \alpha\left(\omega_x', \omega_y'\right)\right] &= \frac{1}{N^2}\sum_i \sum_j \mathrm{Cov}\left[\alpha_i(\omega_x, \omega_y), \alpha_j\left(\omega_x', \omega_y'\right)\right] \\
&= \frac{1}{N^2}\sum_i \mathrm{Cov}\left[\alpha_i(\omega_x, \omega_y), \alpha_i\left(\omega_x', \omega_y'\right)\right]
\end{aligned}$$

where the $\alpha_i(.)$ are the contributions of the individual particles to the density, and the simplification in the second line is possible because of the independence of the trajectories. The individual covariances are

$$\begin{aligned}
\mathrm{Cov}\left[\alpha_i(\omega_x, \omega_y), \alpha_i\left(\omega_x', \omega_y'\right)\right] &= \mathrm{E}\left[\alpha_i(\omega_x, \omega_y)\, \alpha_i\left(\omega_x', \omega_y'\right)\right] \\
&\quad - \mathrm{E}[\alpha_i(\omega_x, \omega_y)]\, \mathrm{E}\left[\alpha_i\left(\omega_x', \omega_y'\right)\right]
\end{aligned}$$

and

$$\mathrm{E}\left[\alpha_i(\omega_x, \omega_y)\, \alpha_i\left(\omega_x', \omega_y'\right)\right] = \frac{1}{2\pi|V_y + R_y|^{1/2}} \exp\Big[-\frac{1}{2} t_y^{\mathrm{T}} (V_y + R_y)^{-1} t_y \Big]$$

where

$$t_y = \begin{pmatrix} \omega_y \\ \omega'_y \end{pmatrix} \quad V_y = \begin{pmatrix} v_y(\omega_x) & c_y(\omega_x, \omega'_x) \\ c_y(\omega'_x, \omega_x) & v_y(\omega'_x) \end{pmatrix} \qquad R_y = \begin{pmatrix} \eta_y^2 & 0 \\ 0 & \eta_y^2 \end{pmatrix}$$

with V_y describing the correlation between the processes at different points, and R_y describing the contribution of the smoothing parameters of the Gaussian kernels. We can do exactly the same calculation for the vertical direction.

Combining the components in both the horizontal and vertical directions, we define the corresponding full random process as

$$\alpha(\omega_x, \omega_y, \omega_z) = \alpha_y(\omega_x, \omega_y)\, \alpha_z(\omega_x, \omega_z) \ .$$

Assuming that the horizontal and vertical driving processes are uncorrelated, the expectation of this process is then

$$\begin{aligned} b(\omega) &= \mathrm{E}[\alpha(\omega_x, \omega_y, \omega_z)] \\ &= \mathrm{E}[\alpha_y(\omega_x, \omega_y)]\, \mathrm{E}[\alpha_z(\omega_x, \omega_z)] \\ &= \frac{1}{\sqrt{2\pi(\eta_y^2 + v_y(\omega_x))}} \exp\Big[-\frac{\omega_y^2}{2(\eta_y^2 + v_y(\omega_x))}\Big] \\ &\quad \times \frac{1}{\sqrt{2\pi(\eta_z^2 + v_z(\omega_x))}} \exp\Big[-\frac{\omega_z^2}{2(\eta_z^2 + v_z(\omega_x))}\Big] \end{aligned}$$

and its covariance is

$$\begin{aligned} c(\omega, \omega') &= \mathrm{Cov}\left[\alpha(\omega_x, \omega_y, \omega_z), \alpha\left(\omega'_x, \omega'_y, \omega'_z\right)\right] \\ &= \frac{1}{N}\Bigg[\frac{1}{2\pi|V_y + R_y|^{1/2}} \exp\Big[-\frac{1}{2} t_y^{\mathrm{T}} (V_y + R_y)^{-1} t_y\Big] \\ &\quad \times \frac{1}{2\pi|V_z + R_z|^{1/2}} \exp\Big[-\frac{1}{2} t_z^{\mathrm{T}} (V_z + R_z)^{-1} t_z\Big] - b(\omega)\, b(\omega')\Bigg] \end{aligned}$$

where V_z, R_z and t_z are defined in the same way as their counterparts in the horizontal plane. It would be possible to introduce additional correlations into the process by assuming a two-dimensional driving fractional Brownian motion in the horizontal and vertical planes; however, doing so would require the evaluation of a 4-dimensional Gaussian pdf when computing the covariance function, which would make the evaluation of $c(\omega, \omega')$ much slower, so we use the above, separable form.

The functions $b(.)$ and $c(.,.)$ have been designed so that the process which they

describe reproduces the attractive properties of the plume; the expected concentration of particulate matter has a Gaussian shape, with a spread which increases as a function of the distance downwind of the source, the correlations between concentration values at different points are derived by considering the likely behaviour of packets of gas under transport by the wind, and the magnitude of the marginal variation at a given point decreases exponentially to zero as we move away from the centre of the plume. These functions will therefore form the basis of our description of the system.

Wind drift When building this model, if we modify the driving process slightly, we can allow for the effect of local deviations in the wind field on the resulting shape of the plume; we introduce a wind drift term into the process, altering the expectation to

$$\mathrm{E}\left[\chi_y\left(\omega_x\right)\right] = \mu_y\left(\omega_x\right)$$

where $\mu_y\left(0\right) = 0$, and

$$\mu_y\left(\omega_x + h_x\right) = \mu_y\left(\omega_x\right) + \int_{\omega_x}^{\omega_x + h_x} \xi_y\left(\zeta\right) d\zeta$$

and where the function $\xi_y\left(\omega_x\right)$ is the effect of the wind perpendicular to the prevailing direction at downwind location ω_x.

The function $\mu_y\left(.\right)$ describes the total effect of wind drifts as a function of downwind distance ω_x, and $\xi_y\left(.\right)$ is a known function describing deviations from the trajectory of the wind in the horizontal direction (with a similar definition for $\xi_z\left(.\right)$). If we compute the mean and covariance functions in the same way as above, we obtain similar Gaussian forms which have undergone a location shift to be centred around

the wind drift value at ω_x

$$
\begin{aligned}
b_w(\omega) &= \frac{1}{\sqrt{2\pi(\eta_y^2 + v_y(\omega_x))}} \exp\Big[-\frac{(\omega_y - \mu_y(\omega_x))^2}{2(\eta_y^2 + v_y(\omega_x))}\Big] \\
&\quad \times \frac{1}{\sqrt{2\pi(\eta_z^2 + v_z(\omega_x))}} \exp\Big[-\frac{(\omega_z - \mu_z(\omega_x))^2}{2(\eta_z^2 + v_z(\omega_x))}\Big] \\
c_w(\omega, \omega') &= \frac{1}{N}\Bigg[\frac{1}{2\pi|V_y + R_y|^{1/2}} \exp\Big[-\frac{1}{2}(t_y - m_y(\omega_x))^{\mathrm{T}}(V_y + R_y)^{-1}(t_y - m_y(\omega_x))\Big] \\
&\quad \times \frac{1}{2\pi|V_z + R_z|^{1/2}} \exp\Big[-\frac{1}{2}(t_z - m_z(\omega_x))^{\mathrm{T}}(V_z + R_z)^{-1}(t_z - m_z(\omega_x))\Big] \\
&\quad - b_w(\omega)\, b_w(\omega')\Bigg]
\end{aligned}
$$

where $m_y(\omega_x) = (\mu_y(\omega_x), \mu_y(\omega_x'))^{\mathrm{T}}$ is the vector of wind drift terms at ω_x and ω', with a similar definition for the vertical term.

Modifications for other atmospheric effects The Gaussian plume model frequently includes modifications to take account of the effect of reflection in the atmospheric boundary layer (ABL) and in the ground. These can be introduced into the stochastic model by simply including the relevant terms in the kernel used to smooth the random processes. We do not consider this further here.

5.6.2 Relating simulators

We now use the stochastic process constructed in the previous section to build a series of emulators describing an improving sequence of models for the atmospheric dispersion problem; all simulators that we consider in this section predict the concentration contribution at an individual location, and so all functions are scalars, i.e: $f_1^{(k)}(.) = f^{(k)}(.)$ at all levels k. With reference to the notation from Section 1.1.3, we model the concentration in the presence of a single source with emission rate ψ. The full input set for all simulators is $\theta = \{x, g, \psi, w, \sigma, \rho_a^{(2)}, \rho_a^*, \xi\}$; the parameter $\rho_a^{(2)}$ is an input to all simulators from the second level onwards, and the parameters ρ_a^* and ξ are inputs to the reified simulator.

Level 1: At the first level, our simulator $f^{(1)}(.)$ is simply

$$f^{(1)}(\theta) = \psi \times b(\theta)$$

with the Hurst parameters set to $H_y = H_z = 1$; this gives a simplified version of the standard Gaussian plume, where, using the notation of Section 1.1.2

$$\kappa_y = \sigma_y^2 \omega_x^2 \qquad \kappa_z = \sigma_z^2 \omega_x^2 \ .$$

Because of the simplicity of our simulator, we also specify that

$$g_1^{(1)}(\theta) = \psi \times b(\theta)$$

so that the basis function agrees exactly with the simulator itself; correspondingly, we specify that $\mathrm{E}\left[\beta_1^{(1)}\right] = 1$ and $\mathrm{Var}\left[\beta_1^{(1)}\right] = 0$. To give numerical stability when we perform the full update, we specify that the covariance of the residual at this stage is

$$\mathrm{Cov}\left[r^{(1)}(\theta), r^{(1)}(\theta')\right] = (10^{-12}) \times I(\theta, \theta')$$

where

$$I(\theta, \theta') = \begin{cases} 1 & \text{if } \theta = \theta' \\ 0 & \text{else.} \end{cases}$$

Level 2: We now consider how our model might evolve in the future, and link these beliefs to the initial simulator within the framework outlined in Section 5.2. We expect that our main focus in this initial development stage will be on improving the model's handling of systematic local effects in the concentration at different points on the downwind trajectory; we know that the simple plume model is far too smooth to capture turbulent effects in the atmosphere and local stochastic behaviour of gas particles. Our experience of the use of the simple model also indicates that it fails to capture changes in the wind field along particle trajectories (see, for example, the landfill data shown in Figure 1.1(a)), but we choose to defer developments in this direction until the next stage.

Before we have begun to explore different developmental avenues it is unclear to us whether the improved model will come about through introducing modifications to the existing plume, or through development of a model which solves the governing equations more carefully; it is also unclear to us at this stage exactly how the additional effects which we will include will be parametrised. However, we are willing

to make an approximate specification for our level of uncertainty at each point on the existing plume through use of the basis and covariance functions developed in Section 5.6.1.

We judge that our expectations for the behaviour of such an improved simulator are best described by a scaled mixture of plumes driven by different values of the Hurst parameter; specifically, we choose to use a mixture of two differently shaped plumes, with parameter $\rho_a^{(2)} \in [0,1]$ determining the contribution of each component, and we choose

$$\begin{aligned} g_1^{(2)}(\theta) &= \rho_a^{(2)} \times g_1^{(1)}(\theta) = \rho_a^{(2)} \times b(\theta|H=1) \\ g_2^{(2)}(\theta) &= (1-\rho_a^{(2)}) \times b(\theta|H=0.9) \end{aligned}$$

where the stated value of the Hurst parameter is used in both the horizontal and the vertical directions. We are uncertain about the combination of these basis functions which gives the best fit to our new simulator, and we judge that an appropriate representation of our uncertainty is given by

$$\begin{aligned} \beta_1^{(2)} &= \beta_1^{(1)} + \epsilon_{\beta_1^{(2)}} \\ \beta_2^{(2)} &= \beta_1^{(1)} + \epsilon_{\beta_2^{(2)}} \end{aligned}$$

for uncorrelated stochastic terms $\epsilon_{\beta_1^{(2)}}$ and $\epsilon_{\beta_2^{(2)}}$. Within this framework, we specify that $\mathrm{E}\left[\epsilon_{\beta_1^{(2)}}\right] = \mathrm{E}\left[\epsilon_{\beta_2^{(2)}}\right] = 0$, and that $\mathrm{Var}\left[\epsilon_{\beta_1^{(2)}}\right] = \mathrm{Var}\left[\epsilon_{\beta_2^{(2)}}\right] = (0.05)^2$, and we can then deduce that

$$\begin{aligned} \mathrm{E}\left[\beta_1^{(2)}\right] &= 1 \quad \mathrm{Var}\left[\beta_1^{(2)}\right] = \mathrm{Var}\left[\epsilon_{\beta_1^{(2)}}\right] = (0.05)^2 \\ \mathrm{E}\left[\beta_2^{(2)}\right] &= 1 \quad \mathrm{Var}\left[\beta_2^{(2)}\right] = \mathrm{Var}\left[\epsilon_{\beta_2^{(2)}}\right] = (0.05)^2 \ . \end{aligned}$$

Our beliefs about the remaining systematic behaviour of the improved simulator are then captured through the residual process $r^{(2)}(.)$; our covariance function for this process is simply a scaled version of the covariance function developed in Section 5.1.1

$$\mathrm{Cov}\left[\nu^{(2)}(\theta), \nu^{(2)}(\theta')\right] = v_\nu^{(2)} \times (1-\rho_a^{(2)})(1-\rho_a^{(2)\prime}) \times (\psi\psi') \times c(\theta,\theta')$$

where $v_\nu^{(2)}$ is a marginal standard deviation term, which scales the contribution of the residual to the total variability. We also choose that $\gamma^{(2)}(\theta) = \rho_a^{(2)}$, so that

uncertainty about the first model is propagated appropriately; since the covariance of the first model is essentially zero, however, we obtain no contribution from this stage when we perform the update anyway. The Hurst parameter for the covariance function is set to $H_y = H_z = 0.9$, corresponding to the choice in the second mean function, and the smoothing parameters are set to be $\eta_y = 50$ and $\eta_z = 50$; the marginal variance term is set as $v_\nu^{(2)} = (2 \times 10^{-3})^2$.

Reified simulator: We now consider the form that our reified simulator might take; we consider all of the further improvements that we might make to the model that we propose to build at stage two, if we had access to an large development budget and computer resources. We propose that we would make improvements in two main areas; we would improve the model's representation of wind drift along the trajectory of the plume, and we would introduce a representation of the diffusion of gas particles that takes place upwind of a gas source.

To describe the effect of including wind-drift in our model, we down-weight the effect of the existing basis functions on the reified simulator, and we introduce a third basis function

$$\begin{aligned} g_1^{(3)}(\theta) &= \rho_a^* \times \rho_a^{(2)} \times b(\theta|H=1) \\ g_2^{(3)}(\theta) &= \rho_a^* \times (1-\rho_a^{(2)}) \times b(\theta|H=0.9) \\ g_3^{(3)}(\theta) &= (1-\rho_a^*) \times b_w(\theta|H=0.9) \end{aligned}$$

where $\rho_a^* \in [0,1]$ is an additional scaling parameter which determines the size of the additional global effect introduced by our wind drift modelling. The wind-drift term in the function $b_w(.)$ is chosen to have a piecewise linear form, i.e.

$$\mu_y(\omega_x) = \int_0^{\omega_x} \xi_y(\omega_x)\, d\omega_x$$

where

$$\xi(\omega_x) = \sum_{k=1}^{n_\xi} \xi_{yk} I_k(\omega_x) \qquad I_k(\omega_x) = \begin{cases} 1 & \text{if } t_{k-1} \leq \omega_x < t_k \\ 0 & \text{else} \end{cases}$$

for knots t_k along the prevailing wind direction and constants ξ_{yk} which are added to the model input set. We use the same specification for $\mu_z(\omega_x)$ as a function of

parameters ξ_z.

We judge that much of the behaviour captured by our model at stage 2 will still be present in our reified model, and so we judge that coefficients of the corresponding basis functions will be strongly correlated across stages; we specify that

$$\begin{aligned}\beta_1^{(3)} &= \beta_1^{(2)} + \epsilon_{\beta_1^{(3)}} \\ \beta_2^{(3)} &= \beta_2^{(2)} + \epsilon_{\beta_2^{(3)}}\end{aligned}$$

where the variances of both additional, uncorrelated, mean zero, stochastic terms are set at $\text{Var}\left[\epsilon_{\beta_1^{(3)}}\right] = \text{Var}\left[\epsilon_{\beta_2^{(3)}}\right] = (0.005)^2$, leading to the following prior specifications

$$\begin{aligned}\text{E}\left[\beta_1^{(3)}\right] &= 1 \quad \text{Var}\left[\beta_1^{(3)}\right] = (0.05)^2 + (0.005)^2 \\ \text{E}\left[\beta_2^{(3)}\right] &= 1 \quad \text{Var}\left[\beta_2^{(3)}\right] = (0.05)^2 + (0.005)^2 \ .\end{aligned}$$

We specify our beliefs about $\beta_3^{(3)}$ in relation to our beliefs about $\beta_2^{(2)}$

$$\beta_3^{(3)} = \beta_2^{(2)} + \epsilon_{\beta_3^{(3)}}$$

where $\text{Var}\left[\epsilon_{\beta_3^{(3)}}\right] = 0.05^2$, which, of course, allows us to find an expectation and a variance for this parameter

$$\text{E}\left[\beta_3^{(3)}\right] = 1 \quad \text{Var}\left[\beta_3^{(3)}\right] = (0.05)^2 + (0.05)^2 \ .$$

Using these relationships, we can also find the covariances between the parameters; for example

$$\begin{aligned}\text{Cov}\left[\beta_1^{(3)}, \beta_2^{(3)}\right] &= \text{Cov}\left[\beta_1^{(2)}, \beta_2^{(2)}\right] = 0 \\ \text{Cov}\left[\beta_1^{(3)}, \beta_3^{(3)}\right] &= \text{Cov}\left[\beta_1^{(2)}, \beta_2^{(2)}\right] = 0 \\ \text{Cov}\left[\beta_2^{(3)}, \beta_3^{(3)}\right] &= \text{Var}\left[\beta_2^{(2)}\right] = (0.05)^2 \ .\end{aligned}$$

For the residual process, we specify that $\gamma^*(\theta) = \rho_a^*$, and that the covariance function for the additional residual process $\nu^*(.)$ is

$$\text{Cov}\left[\nu^*(\theta), \nu^*(\theta)\right] = v_\nu^{(3)} \times (1 - \rho_a^*)(1 - \rho_a^{*\prime}) \times (\psi\psi') \times c_w(\theta, \theta') + c_d(\theta, \theta')$$

where $c_w(.,.)$ is the plume covariance function which includes the wind-drift term and $c_d(.,.)$ is a covariance function describing the systematic variation in the concentration caused by diffusion of gas from the source. Again, $v_\nu^{(3)}$ is a marginal standard deviation parameter which scales the effect of the plume covariance term, and we fix $v_\nu^{(3)} = (1 \times 10^{-3})^2$

Uncertain parameter ranges: In order to make predictions for the system, we must now propagate our uncertainty about the model inputs through our beliefs about the reified simulator; first, we divide our full input set into the sets of simulator inputs a and known system inputs b, so that $\theta = \{a, b\}$. When we make observations of the real concentration, we will know the measurement location x and the wind vector w, and so our collection of system inputs is $b = \{x, w\}$; the remainder of the inputs will be unknown at the point of observation, and so we have that $a = \{g, \psi, \sigma, \rho_a^{(2)}, \rho_a^*, \xi\}$.

We assume that each of these uncertain parameters has a uniform distribution over a given range, with these ranges specified as follows

- $g_x \in [0, 100]$, $g_y \in [0, 100]$;
- $\psi \in [0.0798, 3.3991]$;
- $\sigma_y \in [\tan(10), \tan(10)]$, $\sigma_z \in [\tan(10), \tan(10)]$;
- $\rho_a^{(2)} \in [0.7, 0.8]$;
- $\rho_a^* \in [0.8, 0.9]$;
- $\xi_{yk} \in [0, 0]$, $\xi_{zk} \in [0, 0]$.

Under this specification, we believe that there is a source somewhere in the box $[0, 100] \times [0, 100]$, and we are uncertain about its emission rate; we are also uncertain about the settings of the scaling factors $\rho_a^{(2)}$ and ρ_a^* which control the effects of the different simulators on our eventual prediction for the system value. In the decision problem that we will solve, we will be interested in the ability of different experiments and simulator developments to resolve our uncertainty about the source emission rate.

Discrepancy: Having imagined some of the ways in which the model might change during future development stages and the implications of this for our current uncertainties about the future model predictions, the final component of the structure in the graph 5.1 which it remains for us to specify is the discrepancy between our reified simulator and the real atmospheric concentration data. This consists of two components: the systematic discrepancy $\delta(.)$ and the measurement error ϵ.
First, the systematic discrepancy: as usual, we have included any beliefs about the global structure in our beliefs about the models that we will develop, and so we specify that $\mathrm{E}[\delta(b)] = 0$. After propagation of our beliefs about the best input setting of the parameters a^* through our beliefs about the reified simulator, we judge that our beliefs about any remaining discrepancy are best represented by a stationary stochastic process, and so we specify that

$$\mathrm{Cov}[\delta(b), \delta(b')] = v_\delta c_\delta(b, b')$$

where v_δ is a marginal variance parameter governing the amount of additional variability introduced by the discrepancy at all points, and the correlation function is simply a squared exponential

$$c_\delta(b, b') = \exp\Big[-\frac{1}{2}\sum_j \lambda_{b_j}(b_j - b'_j)^2\Big].$$

We fix the marginal variance for the discrepancy to be $v_\delta = (10^{-3})^2$ and the correlation parameters to be $\lambda_{b_j} = 1/(500^2)$ for the remainder of the example.

Model plots: Predictions for each of the simulator models (using the parameter specifications above) are plotted in Figure 5.2. In each of these plots, the source is fixed at $g = (0, 0)$, and has emission rate $\psi = 3.40$; the wind vector is $w = (2, 2)$, and the scaling factors are fixed to $\rho_a^{(2)} = 0.7$ and $\rho_a^* = 0.8$. Observations are constrained to lie in the horizontal box $[1000, 2000] \times [1000, 2000]$, corresponding to the design problem that we will outline in Section 5.6.3. Figure 5.2(a) shows the expected plume $\mathrm{E}_F\big[f^{(1)}(.)\big]$ at the first level, and Figure 5.2(b) shows the corresponding marginal standard deviation $\mathrm{Var}_F\big[f^{(1)}(.)\big]^{1/2}$; at this level, we are certain about the simulator. Figures 5.2(c) and 5.2(d) show the mean and standard deviation surfaces for the simulator $f^{(2)}(.)$; at this level, the shape of the expected plume within

the displayed region has changed, owing to an additional contribution from the basis function $g^{(2)}(.)$ at a different setting of the Hurst parameters $\{H_y, H_z\}$. We are also now uncertain about the simulator values in this region, since we have not collected any runs at this level yet. Figures 5.2(e) and 5.2(f) show our prediction for the reified simulator based on our current beliefs; the shapes of the mean and standard deviation surfaces are largely unchanged from the simulator at the second level, but there is some additional uncertainty introduced through the unknown scaling on the third basis function and the diffusion term in the covariance function.

Figure 5.3 illustrates the inference procedure for this model; two concentration observations are made (corresponding to the black markers in Figures 5.3(a) and 5.3(b)) under two different wind fields, and these are combined to compute adjusted beliefs for the simulator parameters a^* (as outlined in Section 5.3.3).

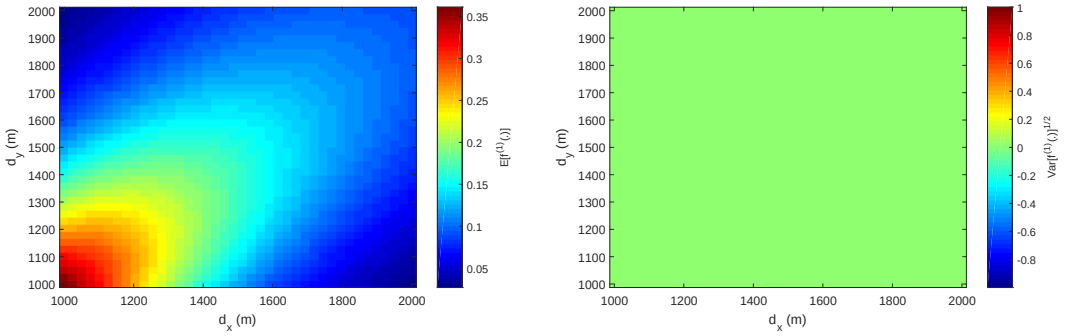
5.6.3 Design problem

Section 5.6.2 sets up the relationship between our current simulator, our current beliefs about future simulators, and the input parameters of the model and the system. In this section, we outline the decision problem that we will solve using observations on the system, and we set up the corresponding design problem that we will approximately solve in Section 5.7.

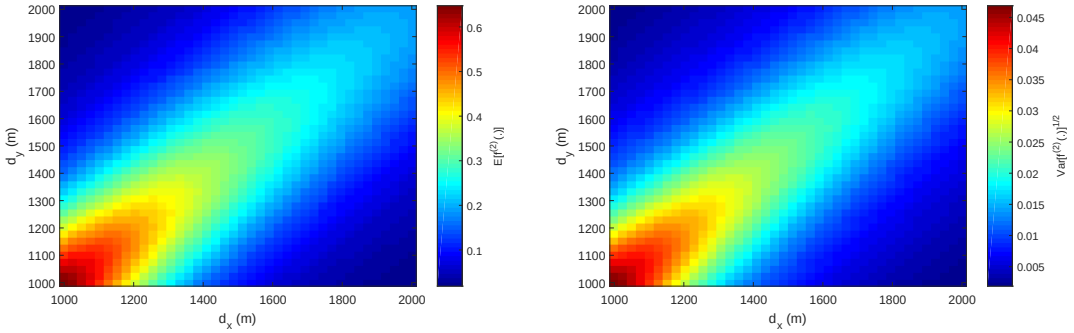
Decision problem As in the decision problem specified in Section 4.1, we choose a weighted quadratic loss function (introduced in Section 3.1.1)

$$\begin{aligned} L(a, q) &= \sum_{k=1}^{n_q} L_k(a_k, q_k) \\ &= \sum_{k=1}^{n_q} \gamma_k(q_k)(q_k - a_k)^2 . \end{aligned}$$

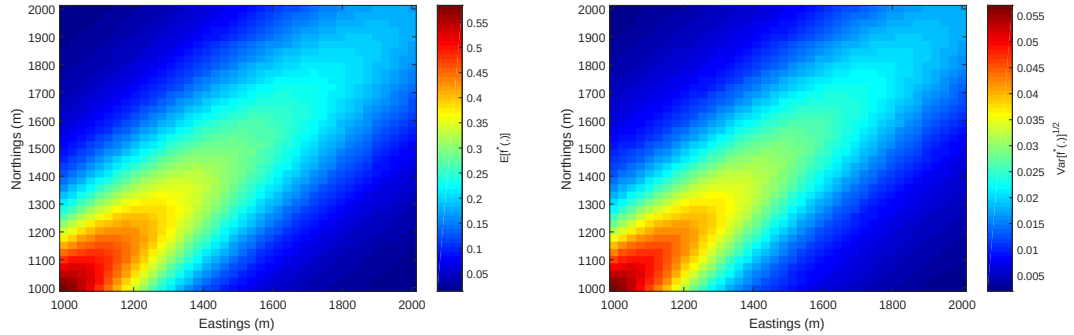
In this problem, the model parameter set which we infer from the data is the best input set for the simulator, with $q = \{g, \psi, \sigma, \rho_a^{(2)}, \rho_a^*, \xi\}$. For the remainder of the example, we assume that our losses will depend only on our beliefs about the source emission rate ψ; we fix $\gamma_2(q_2) = 10^5$, and $\gamma_k(q_k) = 0$ for all $k \neq 2$.



(a) (b)

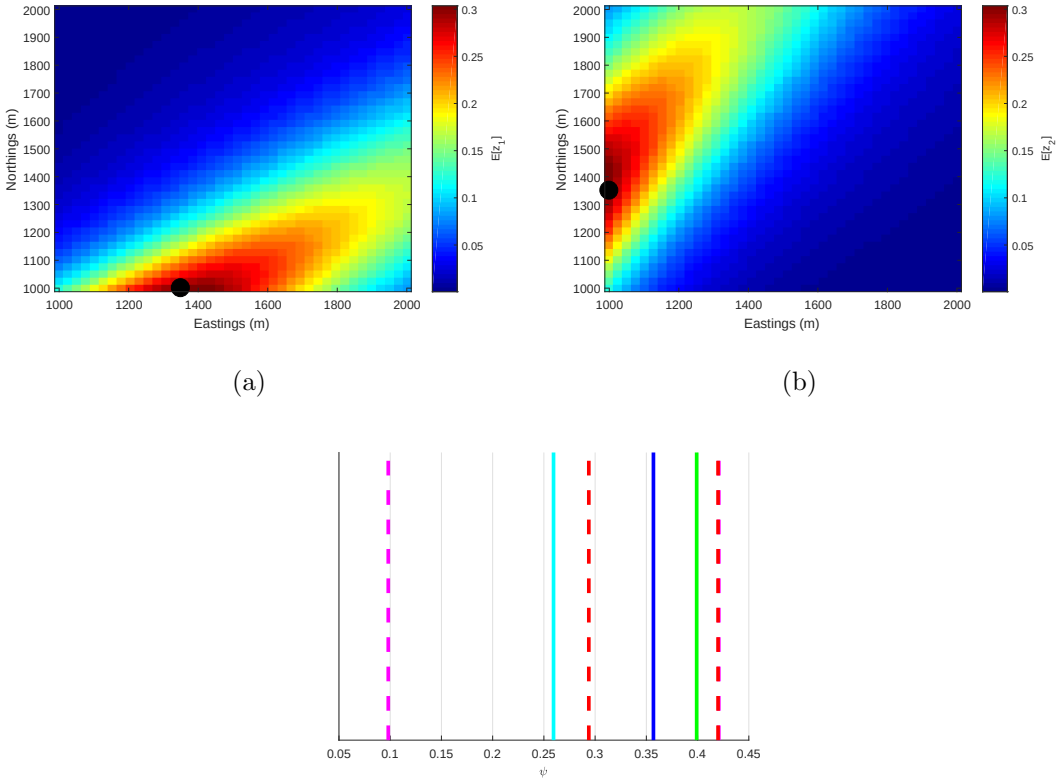


(c) (d)



(e) (f)

Figure 5.2: Plots of the simulator levels: Figures 5.2(a) and 5.2(b) show the mean and standard deviation of the simulator $f^{(1)}$ at the first stage; Figures 5.2(c) and 5.2(d) show our predictive mean and standard deviation surfaces for the simulator $f^{(2)}$, given the uncertainty specification outlined in Section 5.6.2; Figures 5.2(e) and 5.2(f) show our predictive mean and standard deviation surfaces for the reified simulator f^*, again using the uncertainty specification outlined in Section 5.6.2.



(a) (b)

(c)

Figure 5.3: Example inference using the model outlined in Section 5.6.2: the colour scales in Figures 5.3(a) and 5.3(b) show the expected values of the data that we could collect at each of the available design choices under two different sets of wind conditions ($w = [3.15, 2.15]$ in Figure 5.3(a) and $w = [2.15, 3.15]$ in Figure 5.3(b)), and the locations of the the actual observations are shown as black markers. Figure 5.3(c) shows the effect of these two data points on our beliefs about the emission rate: our prior expectation $\mathrm{E}[\psi^*]$ is shown in cyan, with two-standard deviation error bars shown in dashed magenta, and our adjusted mean and error bars are shown in blue and dashed red respectively. The true value of ψ^* used to generate the data is shown in green.

Experimental design problem For this more complex model, we solve a much simpler design problem. There are two field experiments that we may choose to carry out in sequence; in both of them, we may make a single concentration observation using a point-sensor fixed to a location near the ground. Additionally, we specify that we will construct the simulator $f^{(2)}(.)$ between the two experiments, and obtain runs F_2.

In this problem, then, the design locations d_j at each stage (Section 5.4) correspond to the measurement locations x (Section 5.6.2), with $d_j = \{x_{jx}, x_{jy}, x_{jh}\}$; at both stages $j = 1, 2$, we restrict the observations that we can make to a box:

- $d_{jx} \in [1000, 2000]$;
- $d_{jy} \in [1000, 2000]$;
- $d_{jh} \in [0.5, 3]$.

We also specify that the wind field at both stages is subject to uncertainty. We identify the external parameters w_j at each stage (Section 5.4) with the wind field w for each measurement, with $w_j = \{w_{jx}, w_{jy}\}$; at both stages, we assume that the components of the wind vector are independently uniformly distributed, over the following ranges:

- stage 1: $w_{1x} \in [3, 3.3]$, $w_{1y} \in [2, 2.3]$;
- stage 2: $w_{2x} \in [2, 2.3]$, $w_{2y} \in [3, 3.3]$.

We also specify that $\mathrm{Var}[\epsilon_1] = (0.001)^2$ and $\mathrm{Var}[\epsilon_2] = (0.05)^2$, so that the data from the first experiment is measured with greater accuracy than the data from the second. We fix the costs of both experiments to be independent of the design parameter setting, with $c_1(d_1) = 5$ and $c_2(d_2) = 5$.

Simulator design problem Since we have specified that we will definitely construct the simulator $f^{(2)}(.)$ between experiments, and that there are no decisions about the development of this simulator that we need to make, we have that $\phi_{[2]} = \emptyset$ for this problem; this removes the need for us to carry out the optimisation step over simulator design parameters detailed in Section 5.5.5, and means that we simply

compare the expected risk over simulator runs $\bar{t}_2^{(i)}$ with the risk from an immediate decision after the first experiment in order to characterise our approximations $r_1^{(i)}$ to the risk from experimentation at the first stage. We fix the cost of constructing the simulator at stage 2 to be constant, at $c_2^f(\phi_2) = 1$.

5.7 Example: Running the backward induction

Having specified a full set of relationships between the simulators that we might construct and the system (Section 5.6.2) and set out the decision and design problems that we wish to solve for this model (Section 5.6.3), we are in a position to run the approximate backward induction algorithm detailed in Section 5.5. In this section, we provide details of our implementation for this example, and plots of the resulting emulators and design spaces; in Section 5.7.1, we specify the basis and covariance function choices that we make for each of the emulators that we fit, and in Section 5.7.2, we provide details relating to the risk model fitting procedure. In Section 5.8, we consider issues arising from the approximate design procedure, and from this example, and identify areas for future work.

Numerical integration Before we proceed with the backward induction approximation, we must specify our strategy for evaluation of the terminal risks ρ_j^t and ρ_j^f. For the separable weighted quadratic loss that we have chosen, the optimal terminal decision and the risk from this choice are easy to identify in terms of the moments of a general posterior distribution (see Section 3.1.1); however, evaluation of these moments within the framework outlined in Section 5.6.2 is challenging. Under our second-order prior specification, the calculations that we must perform in order to find our second-order adjusted moments $\mathrm{E}_{z_{[j]}}[q]$ and $\mathrm{Var}_{z_{[j]}}[q]$ are presented in Section 5.3.3; however, for our problem, we cannot perform the basis and covariance function integrals algebraically, owing to the complex form of the stochastic process which we have chosen to represent plume behaviour (Section 5.6.1).

We must therefore select a numerical procedure for approximation of these quantities. Our chosen approximation method is as follows:

- the full covariances (5.3.6) are approximated by evaluating only (5.3.7) and the first term from the expression (5.3.8); the remaining terms contain the integrals which are the most computationally-intensive to evaluate numerically, and we can get a reasonable approximation to the full covariance by simply ignoring them;

- the remaining basis functions are evaluated by sampling the integrand at a space-filling sample of 2500 points and computing the sample average.

The level of variation in the risks ρ_j^t and ρ_j^f induced by variation in the sample of points used to evaluate the integrals in the expression for the moments (5.3.5) and (5.3.6) is judged to be negligible in comparison to the variation in the risk across its design space, and so we proceed with this as an acceptable and computationally cheap way of approximating the risks throughout the remainder of the procedure.

5.7.1 Basis and covariance functions

In this section, we provide details of the basis and covariance function choices that we make for all emulators fitted in Section 5.7.2.

Stage 2: experimental risk For the experimental risk model $r_2^{(1)}$ at the final stage of the calculation, we use a basis function specification analogous to the one used in the linear model example (Section 3.4.5) and the atmospheric dispersion model example (Section 4.2). Our basis consists of an intercept $h_{21}^{(1)} = 1$, and a terminal risk function with a modified input set

$$h_{22}^{(1)}\left(z_{[2]}, w_{[2]}, d_{[2]}, F_{[2]}, \phi_{[2]}\right) = \rho_2^t\left[\tilde{z}_{[2]}, \tilde{w}_{[2]}, d_{[2]}, F_{[2]}, \phi_{[2]}\right]$$

where, as usual, $\tilde{z}_{[2]} = \{z_1, \mathrm{E}_{z_{[1]}}\left[z_2|\tilde{w}_{[2]}, \ldots\right]\}$ and $\tilde{w}_{[2]} = \{w_1, \mathrm{E}\left[w_2\right]\}$. This choice of basis functions allows the regression surface to explain a large proportion of the variation in the risk, while ensuring that we can integrate the basis functions as required when computing beliefs about $\bar{r}_2^{(1)}$.

For the covariance function, we choose a separable squared exponential form, as

follows

$$\text{Cov}\left[u_2^{(1)}(.), u_2^{(1)}(.')\right] = \eta_2^{(1)}\Big[\prod_{k=1}^{2}\big[\prod_{p=1}^{2} c\left(w_{kp}, w'_{kp}|\lambda_{w_p}\right)\big]\big[\prod_{p=1}^{3} c\left(d_{kp}, d'_{kp}|\lambda_{d_p}\right)\big]\Big]$$

where $c(.,.|\lambda)$ is a squared exponential correlation function (Appendix B.1) with correlation parameter λ, and $\eta_2^{(1)}$ is the marginal variance of the process. As discussed in Section 5.5.3, the correlation parameters are determined through leave-one-out cross-validation.

To compute the moments of $\bar{r}_2^{(1)}$ (as outlined in Sections 5.5.3, 2.4.1), we must compute integrals of the basis and covariance functions. Because of the choices that we have made for the basis functions, we have that $\bar{h}_{2p}^{(1)} = h_{2p}^{(1)}$ for $p = 1, 2$. Integrating the covariance function once with respect to $p\left(z_2|z_{[1]}, w_{[2]}, \dots\right)$ and $p\left(w_2|w_{[1]}, d_{[1]}\right)$, we find that

$$\begin{aligned}\text{Cov}\left[\bar{u}_2^{(1)}(.), u_2^{(1)}(.')\right] = \eta_2^{(1)}\Big[\prod_{k=1}^{2}\big[\prod_{p=1}^{3} c\left(d_{kp}, d'_{kp}|\lambda_{d_p}\right)\big]\Big] \\ \times\big[\prod_{p=1}^{2} c\left(w_{1p}, w'_{1p}|\lambda_{w_p}\right)\bar{c}\left(w'_{2p}|\lambda_{w_p}\right)\big]\end{aligned}$$

and integrating a second time, we have

$$\begin{aligned}\text{Cov}\left[\bar{u}_2^{(1)}(.), \bar{u}_2^{(1)}(.')\right] = \eta_2^{(1)}\Big[\prod_{k=1}^{2}\big[\prod_{p=1}^{3} c\left(d_{kp}, d'_{kp}|\lambda_{d_p}\right)\big]\Big] \\ \times\big[\prod_{p=1}^{2} c\left(w_{1p}, w'_{1p}|\lambda_{w_p}\right)\bar{\bar{c}}\left(\lambda_{w_p}\right)\big]\end{aligned}$$

where $\bar{c}\left(.|\lambda_{w_p}\right)$ and $\bar{\bar{c}}\left(\lambda_{w_p}\right)$ are squared exponential covariance functions integrated with respect to the uniform distributions $p\left(w_{2p}\right)$, $p = 1, 2$. Details of these calculations are supplied in Appendix B.1.2.

Stage 2: simulator risk For the simulator risk at stage $j = 2$, we again fix $\hat{h}_{21}^{(1)} = 1$, and we specify that

$$\begin{aligned}&\hat{h}_{22}^{(1)}\left(z_{[1]}, w_{[1]}, d_{[1]}, \tilde{F}_{[2]}, \phi_{[2]}\right) \\ &= \min\left[\rho_2^f\left[z_{[1]}, w_{[1]}, d_{[1]}, \tilde{F}_{[2]}, \phi_{[2]}\right], \text{E}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\left[z_{[1]}, w_{[1]}, \bar{d}_{[2]}, \tilde{F}_{[2]}, \phi_{[2]}\right]\right] + c_2\left(\bar{d}_2\right)\right]\end{aligned}$$

where $\tilde{F}_{[2]} = \mathrm{E}\left[F_2\right]$ and $\bar{d}_{[2]} = \{d_1, \bar{d}_2\}$, with $\bar{d}_2 = [1000, 1300, 2]$ fixed to a design setting which we believe is likely to be optimal for a wide range of the other parameter settings.

For the residual process, we choose another separable squared exponential covariance function

$$\mathrm{Cov}\left[\hat{u}_2^{(1)}(.), \hat{u}_2^{(1)}(.')\right] = \eta_2^{(1)}\Big[\prod_{k=1}^{2}\big[\prod_{p=1}^{2} c\left(w_{kp}, w'_{kp}|\lambda_{w_p}\right)\big]\big[\prod_{p=1}^{3} c\left(d_{kp}, d'_{kp}|\lambda_{d_p}\right)\big]\Big] \times c\left(e_{F_2}, e'_{F_2}|\lambda_{e_F}\right)$$

where $e_{F_2} = E_{F_2}^{\mathrm{T}} F_2$, and E_{F_2} is the eigenvector corresponding to the largest eigenvalue of $\mathrm{Var}\left[F_2\right]$. e_{F_2} is the projection of the simulator runs F_2 on to the direction which explains the largest proportion of their variation; we believe that this will be a low-dimensional summary of the F_2 which will enable us to explain the variability in the risk due to the simulator runs. $\eta_2^{(1)}$ is the marginal variance of the residual process, and $\{\lambda_d, \lambda_w, \lambda_{e_F}\}$ are the correlation parameters; again, the correlation parameters are determined when fitting the emulator by means of a leave-one-out cross-validation scheme.

In order to compute the moments of the expected simulator risk $\bar{t}_2^{(1)}$, we must integrate the basis and covariance functions with respect to $p\left(F_2|\phi_2\right)$. Again, we have selected basis functions which have no dependence on F_2, and so we have that $\bar{\hat{h}}_{2p}^{(1)} = \hat{h}_{2p}^{(1)}$ for $p = 1, 2$. For the covariance functions, we have that

$$\mathrm{Cov}\left[\bar{\hat{u}}_2^{(1)}(.), \hat{u}_2^{(1)}(.')\right] = \eta_2^{(1)}\Big[\prod_{k=1}^{2}\big[\prod_{p=1}^{2} c\left(w_{kp}, w'_{kp}|\lambda_{w_p}\right)\big]\big[\prod_{p=1}^{3} c\left(d_{kp}, d'_{kp}|\lambda_{d_p}\right)\big]\Big] \times \bar{c}\left(e'_{F_2}|\lambda_{e_F}\right)$$

and

$$\mathrm{Cov}\left[\bar{\hat{u}}_2^{(1)}(.), \bar{\hat{u}}_2^{(1)}(.')\right] = \eta_2^{(1)}\Big[\prod_{k=1}^{2}\big[\prod_{p=1}^{2} c\left(w_{kp}, w'_{kp}|\lambda_{w_p}\right)\big]\big[\prod_{p=1}^{3} c\left(d_{kp}, d'_{kp}|\lambda_{d_p}\right)\big]\Big] \times \bar{\bar{c}}\left(\lambda_{e_F}\right)$$

where, in this instance, $\bar{c}\left(.|\lambda_{e_F}\right)$ and $\bar{\bar{c}}\left(\lambda_{e_F}\right)$ are squared exponential covariance functions integrated with respect to the Gaussian $p\left(e_{F_2}|\mu_{e_{F_2}}, V_{e_{F_2}}\right)$; see Appendix

B.1.2 for details of this calculation. The parameters of this distribution are fixed to $\mu_{e_{F_2}} = E_{F_2}^{\mathrm{T}} \mathrm{E}\left[F_2\right]$ and $V_{e_{F_2}} = E_{F_2}^{\mathrm{T}} \mathrm{Var}\left[F_2\right] E_{F_2}$, where $\mathrm{E}\left[F_2\right]$ and $\mathrm{Var}\left[F_2\right]$ are computed as outlined in Section 5.3.2.

Stage 1: experimental risk For the experimental risk at stage $j = 1$, we use the same specification as at the final stage. For the basis functions, we choose $h_{11}^{(1)} = 1$, and

$$h_{12}^{(1)}\left(z_{[1]}, w_{[1]}, d_{[1]}, F_{[1]}, \phi_{[1]}\right) = \rho_1^t\left[\tilde{z}_{[1]}, \tilde{w}_{[1]}, d_{[1]}, F_{[1]}, \phi_{[1]}\right]$$

where $\tilde{z}_{[1]} = \mathrm{E}\left[z_1 | \tilde{w}_{[1]}, d_{[1]}, \ldots\right]$ and $\tilde{w}_{[1]} = \mathrm{E}\left[w_1\right]$.

For the residual process, we use a squared exponential covariance function

$$\mathrm{Cov}\left[u_1^{(1)}(.), u_1^{(1)}(.')\right] = \eta_1^{(1)}\left[\prod_{p=1}^{2} c\left(w_{1p}, w'_{1p} | \lambda_{w_p}\right)\right]\left[\prod_{p=1}^{3} c\left(d_{1p}, d'_{1p} | \lambda_{d_p}\right)\right]$$

where again, $\eta_1^{(1)}$ is the marginal variance of the process, and the parameters $\{\lambda_w, \lambda_d\}$ are determined using leave-one-out cross-validation when fitting the emulator. For the expected experimental risk $\bar{r}_1^{(1)}$ at this stage, the basis and covariance functions are integrated in exactly the same way as at the second stage.

5.7.2 First wave

In this section, we outline details of each of the emulators that we fit when carrying out the procedure from Section 5.5. After fitting each emulator, we check the fit using an additional set of risk evaluations, ensuring that fewer than 5% of these data lie outside three-standard deviation error bars for emulator predictions at the corresponding inputs.

Stage $j = 2$: experimental risk Our first task is to approximate the risk ρ_2 at the final stage of the problem; the basis and covariance functions that we use are discussed in Section 5.7.1. This being the final stage, the risk is $\rho_2 = \rho_2^t$, and so generating risk evaluations is simple (as discussed in Section 3.4.4). We generate 200 risk evaluations for the initial regression, 400 for the joint regression-residual update and 100 for post-fit model checking. Performing the initial regression, we fix the prior specification for the regression parameters; we set $\mathrm{E}\left[\alpha_{21}^{(1)}\right] = 2.17$ and

$\mathrm{E}\left[\alpha_{22}^{(1)}\right] = 0.950$, with $\mathrm{Var}\left[\alpha_{21}^{(1)}\right] = (1.5 \times 10^{-3})^2$ and $\mathrm{Var}\left[\alpha_{22}^{(1)}\right] = (7.7 \times 10^{-6})^2$. Using the residuals from this regression, we set $\mathrm{Var}\left[u_2^{(1)}\right] = \eta_2^{(1)} = (8.11)^2$.

Next, the correlation parameters of the residual covariance function are determined through leave-one-out cross-validation. Using the 400 risk evaluations generated for the joint update, we leave out each point in turn and predict it using the fit to the remainder, for a particular setting of the correlation parameters. We compute the sum of log-Gaussian predictive likelihoods for all points for a Latin hypercube of 2000 different correlation parameter settings, and fix the correlation parameters for the fit to the setting from this collection which performs best.

We complete the emulator update by computing and storing the moments of the data $R_2^{(1)}$. We plot the expectation $\mathrm{E}_{R_2^{(1)}}\left[r_2^{(1)}\right]$ and standard deviation $\mathrm{Var}_{R_2^{(1)}}\left[r_2^{(1)}\right]^{1/2}$ for a grid of points in (d_{2x}, d_{2y})-space in Figures 5.4(a) and 5.4(b) respectively, for fixed settings of the remaining parameters.

Stage $j = 2$: simulator risk Having modelled the experimental risk, we turn our attention to the simulator risk. As discussed in Section 5.5.4, we are uncertain about the optimal experimental risk $s_2^{(1)}$ at this stage, and we must take account of this when generating risk evaluations for fitting the model $t_2^{(1)}$ to the simulator risk at this stage. The moments $\mathrm{E}\left[T_{2k}^{(1)}\right]$ and $\mathrm{Cov}\left[T_{2k}^{(1)}, T_{2l}^{(1)}\right]$ are characterised by sampling the expression (5.5.25), using the moments $\mathrm{E}\left[s_2^{(1)}[.]\right]$ and $\mathrm{Cov}\left[s_2^{(1)}[.], s_2^{(1)}[.']\right]$ obtained as outlined in Section 5.5.3 (and Section 3.4.8). As in the example presented in Section 4.2, we use a single candidate design to approximate the moments of $s_2^{(1)}$ at each setting of its inputs; the expectation (3.4.17) and the second term of the covariance (3.4.18) are evaluated at this single setting, and the first term of (3.4.18) is fixed to be constant across the input space, with $\mathrm{Var}\left[\mathrm{E}_{R_2^{(1)}}\left[\bar{r}_2^{(1)}\left[\tilde{d}_2\right]\right] + c_2\left(\tilde{d}_2\right)\right] = (3.16)^2$ fixed by sampling 20 candidate designs at each of 10 different input settings, and all covariances between different input settings set to zero.

Using this specification, we generate the risk evaluations for the fit; we generate 100 evaluations for the initial regression, 200 for the joint regression-residual update and 100 for post-fit model checking. The basis and covariance functions that

we use for our emulator are discussed in Section 5.7.1. Performing the regression, we fix $\mathrm{E}\left[\hat{\alpha}_{21}^{(1)}\right] = 3.96$ and $\mathrm{E}\left[\hat{\alpha}_{22}^{(1)}\right] = 0.979$, with $\mathrm{Var}\left[\hat{\alpha}_{21}^{(1)}\right] = (3.1 \times 10^{-3})^2$ and $\mathrm{Var}\left[\hat{\alpha}_{22}^{(1)}\right] = (2.3 \times 10^{-5})^2$, and using the residuals from this fit, we fix $\eta_2^{(1)} = (10.7)^2$. The correlation parameters are then fixed through leave-one-out cross-validation, where we compare a Latin hypercube of 2000 potential correlation parameter settings.

The moments of the fitted model are plotted in Figure 5.4: Figure 5.4(c) shows the adjusted expectation $\mathrm{E}_{T_2^{(1)}}\left[t_2^{(1)}\right]$ and Figure 5.4(d) the adjusted standard deviation $\mathrm{Var}_{T_2^{(1)}}\left[t_2^{(1)}\right]^{1/2}$ for a grid of points in (d_{1x}, d_{1y})-space.

Stage $j = 1$: experimental risk Lastly, we fit an model $r_1^{(1)}$ for the experimental risk ρ_1 at the first stage. There are no simulator design parameters to be optimised over in this example, and so the candidate simulator design sampling procedure outlined in Section 5.5.5 is not required. The moments $\mathrm{E}\left[T_{2k}^{(1)}\right]$ and $\mathrm{Cov}\left[T_{2k}^{(1)}, T_{2l}^{(1)}\right]$ are computed for any input setting by sampling the expression 5.5.24 using a Gaussian distribution for $t_2^{(1)}$, characterised by the moments $\mathrm{E}\left[t_2^{(1)}[.]\right]$ and $\mathrm{E}\left[t_2^{(1)}[.], t_2^{(1)}[.']\right]$. We generate 200 risk evaluations for the initial regression, 400 for the joint regression-residual update, and 100 for post-fit model checking. Performing the initial regression, we fix $\mathrm{E}\left[\alpha_{11}^{(1)}\right] = 37.1$ and $\mathrm{E}\left[\alpha_{12}^{(1)}\right] = 0.419$, with $\mathrm{Var}\left[\alpha_{11}^{(1)}\right] = (1.5 \times 10^{-3})^2$ and $\mathrm{Var}\left[\alpha_{12}^{(1)}\right] = (6.2 \times 10^{-6})^2$, and using the residuals from this fit, we fix $\eta_1^{(1)} = (11.3)^2$. For the leave-one-out cross-validation, we test a design of 2000 candidate design settings. The fitted model is illustrated in Figure 5.4; our adjusted expectation $\mathrm{E}_{R_1^{(1)}}\left[r_1^{(1)}\right]$ is shown in Figure 5.4(e), and our adjusted standard deviation $\mathrm{Var}_{R_1^{(1)}}\left[r_1^{(1)}\right]^{1/2}$ in Figure 5.4(f). Both quantities are plotted for a range of settings of (d_{1x}, d_{1y}), for fixed values of the remaining risk inputs.

Stopping Since we have already constructed the simulator at the first stage, the model $r_1^{(1)}$ for the experimental risk at stage 1 is the final model that we must fit; we now perform the analysis detailed in Section 5.5.6 (and Section 3.4.10) to determine what we should now do. First, we interrogate the expected risk surface $\mathrm{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}[d_1]\right] + c_1(d_1)$ at a Latin hypercube of 2000 points in order to approximately identify the design setting which minimises this quantity. On

this basis, we fix $\hat{d}_1 = [1990, 1360, 1.90]$, and our expected risk at this point is $\mathrm{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\left[\hat{d}_1\right]\right] + c_1\left(\hat{d}_1\right) = 66.73$, with $\mathrm{Var}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\left[\hat{d}_1\right]\right] = (1.97)^2$. Comparing this with the risk $\rho_0^t = 650.68$ from an immediate decision, we see that, based on this wave of analysis, we should perform the first experiment at design setting $\hat{d}_1$.

To provide guidance when making a choice between performing the first experiment at this setting and commissioning a further wave of analysis, we compute the expected value of perfect information for the risk calculation. By approximating the second term in the expression (3.4.19) by sampling 200 candidate designs, we find that $v_1^{(1)} = 0.97$. When making decisions about further waves of analysis, this EVPI value should be compared with the cost of another wave of analysis: if we expect to be able to achieve a sufficiently large reduction in the EVPI, then we should pay for another wave; otherwise, we should just proceed optimally based on our current beliefs about the risk. For he purposes of illustration, however, we stop here.

5.8 Discussion

The algorithm presented in Section 5.5 for sequential experimental design where model development is envisaged is very closely related to the one presented in Section 3.4; many of the comments made about the strengths and weaknesses of the original algorithm in Section 4.3 therefore also apply to this algorithm as well. The bulk of the computational effort involved in both algorithms is required to approximately sample from the minimum of the risk in design space for different settings of the design and data parameters at previous stages; any future developments in this area would therefore have the greatest impact on the computational complexity of the algorithm, and on its ability to handle more complex problems.

In this particular instance, an additional factor affecting the ability of the algorithm to generate good designs is the choice that we make to neglect the effect of the design problem for the model developments. In cases where the models $f^{(j)}(.)$ are relatively fast to run, have low-dimensional input spaces, and where we can obtain large numbers of runs on the model, this approximation is unlikely to make a large difference to the design space identified by the algorithm; however, in the (common)

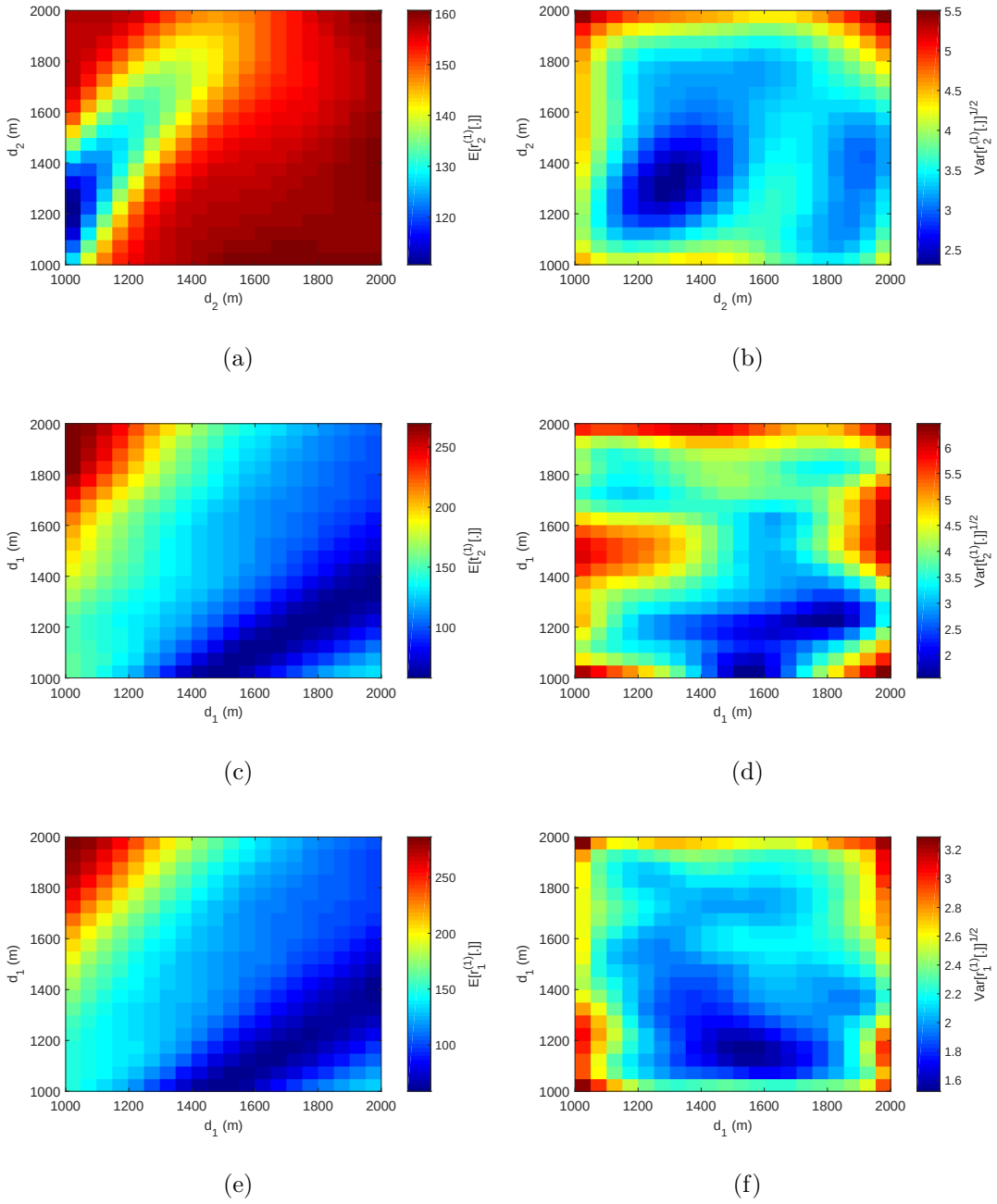


Figure 5.4: Plots of the fitted emulators from the first wave of the analysis: Figure 5.4(a) shows the mean level $\mathrm{E}_{R_2^{(1)}}\left[r_2^{(1)}[.]\right]$ of the emulator for the experimental risk, and Figure 5.4(b) shows the corresponding standard deviation, $\mathrm{Var}_{R_2^{(1)}}\left[r_2^{(1)}[.]\right]^{1/2}$; Figures 5.4(c) and 5.4(d) show the mean $\mathrm{E}_{T_2^{(1)}}\left[t_2^{(1)}[.]\right]$ and $\mathrm{Var}_{T_1^{(2)}}\left[t_2^{(1)}[.]\right]^{1/2}$ of the emulator fitted to the simulator risk; Figures 5.4(e) and 5.4(f) show the corresponding moments for the $r_1^{(1)}[.]$.

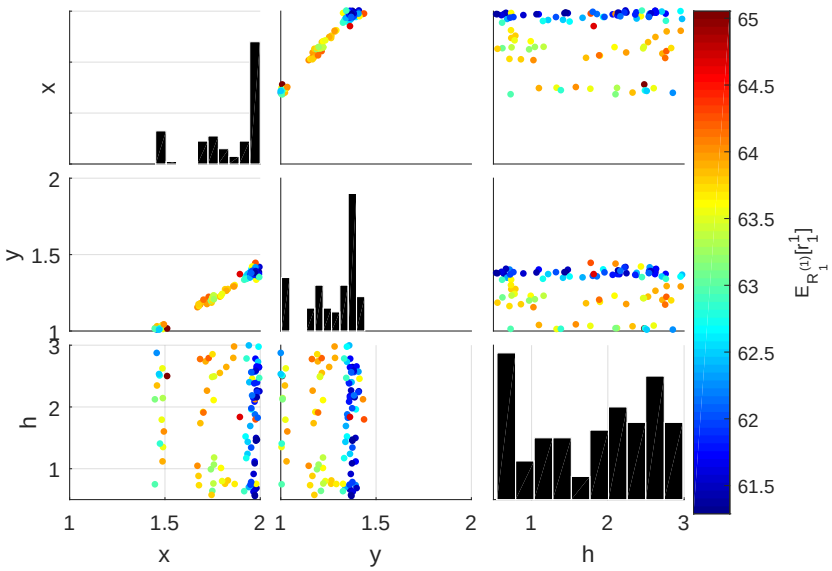


Figure 5.5: Set of candidate designs $\tilde{d}_1$ generated from the emulator $\bar{r}_1^{(1)}$ as detailed in Section 3.4.8; the colour scale indicates the expected risk $\mathrm{E}_{R_1^{(1)}}\left[\bar{r}_1^{(1)}\left[\tilde{d}_1\right]\right] + c_1\left(\tilde{d}_1\right)$ at each point.

situation where the model is high-dimensional and slow to run, we may only have access to a small number of runs, and the design choices that we make for them may have a big effect on the result. If it is also the case that it is much cheaper to generate additional model runs than to perform experiments on the real system, then it may be possible for us to reduce the cost of the overall procedure by doing a better job of resolving uncertainty in the simulator (using a better design), and collecting fewer observations on the system. If we were to take account of the design selections for the model development stages, then it is likely that the optimal designs that we obtain for the model will be strongly influenced by the designs that we choose for the system, as the system designs will also be influenced by the model designs.

Chapter 6

Bayes linear numerical modelling

Thus far in this thesis, we have considered stochastic Bayesian representations of complex deterministic simulators, and stochastic descriptions of the relationships between these simulators and the systems that they represent. For the purposes of our previous analyses, these complex, deterministic simulators are treated as 'black box' functions; that is, no attempt is made to investigate the internal structure of the simulator, it is simply run at a handful of known input settings, and the resulting output is used to fit a statistical model as a fast, cheap representation of its behaviour.

In practice, many such 'black box' simulators are pieces of code which solve ordinary or partial differential equation (ODE or PDE) systems for particular input settings; in the majority of cases, these systems of differential equations are solved numerically, because an analytic solution is not available. For an ODE system, a numerical solver approximates the solution trajectory at a set of knots by recursively passing the current state through a transfer function based on the form of the differential equations; numerical PDE solvers work in a similar way, imposing a mesh in both space and time and producing a system of equations which relates sets of discrete spatial unknowns across time-steps. Using a numerical solution to the PDE allows us to make progress with a problem where before this was impossible, at the cost of introducing a discrepancy between the real solution that we implicitly specified through our choice of differential equation model and the approximate solution that we obtain using our solver.

Knowledge of the underlying structure presents us with the opportunity to tailor our model; instead of just directly relating simulator inputs and outputs, we can carry out our modelling on a much 'lower' level by making an uncertainty specification for each of the input components and then tracking this uncertainty through each step of the numerical solver. Modelling in this way would allow us to develop a much more detailed representation of our uncertainty about the model output across different settings of the model input than we would have been able to achieve simply by modelling the simulator output directly. Additionally, modelling solution uncertainty in this way allows us to explicitly account for the discrepancy between our numerical solution and the real solution to the differential equation system as it builds up, and to deduce its implications for our uncertainty about the solution surface. Much research attention has recently been paid to this problem.

In this chapter, we propose Bayesian modelling frameworks for both the ODE and PDE problems. In Section 6.1, we consider standard numerical solvers for ODE systems, and suggest a modelling framework which accounts for the structure of the numerical discrepancy. We then illustrate the framework through application to two examples; in Section 6.1.7, we consider a simple problem, in which an ODE determines the trajectory of a projectile, and in Section 6.2, we consider a more complex example, in which a system of coupled ODEs is used to represent the interaction of a set of ringing bells and the church bell tower in which they are mounted. In Section 6.3, we consider the PDE problem; we discuss some common numerical schemes for PDE, and we consider the structure of the numerical discrepancy introduced through use of such a solver. We develop a similar framework for handling this uncertainty, and illustrate this through application to a simple, one-dimensional example in Section 6.3.4. We discuss the work in this chapter and consider areas for future research in Section 6.4.

6.1 Ordinary differential equations

An ordinary differential equation (ODE) model implicitly defines the behaviour of a function through specification of its derivatives with respect to a single variable

(usually time). Such models are ubiquitous in mathematical modelling:

- In classical mechanics, Newton's second law is used to relate the forces acting on a system to the accelerations of its components; this results in systems of second-order ODEs which must then be integrated (numerically or algebraically) in time in order to obtain descriptions of the positions and velocities of the components (see, for example, Kibble and Berkshire [2004]);
- In climate modelling, ODE systems are sometimes used as simple representations of the rates of change of system properties (for example, temperature, salinity, pressure) as a function of current state and parameters (see, for example, Zickfeld et al. [2004]).

If a set of ODEs is to be used as a model for a system, then these must be solved in order to generate a prediction for the system. There are a number of different strategies which can be used to obtain an analytic solution to the equations; however, these strategies can only be used in a narrow range of situations and so, for most problems, numerical methods must be used to approximate the solution.

Numerical solution schemes for ODEs work by imposing a grid on the solution input domain and then relating the solution at each time-step to the solution at the previous time-step by approximating the integral of the time-derivative function between these points. A variety of different such schemes are available, each of which relates the approximation at one time-step to that at the next through some functional of the derivative function and the grid properties. In choosing to use a particular numerical scheme, we are introducing a particular discrepancy between the approximate solution that we obtain and the unknown, true solution that we envisaged when we first specified the ODE system. When relating the solution to the system that it is meant to represent, we must account for this numerical discrepancy, alongside all of the usual sources of uncertainty (for example, best input uncertainty, and uncertainty about the discrepancy between the underlying model and the system).

The remainder of this section is structured as follows. In Section 6.1.1, we outline the general form for an ODE solver, and consider some of the most commonly-used types;

then, in Section 6.1.2, we specify a general form for the numerical discrepancy, and consider its leading-order behaviour. In Section 6.1.3, we consider the different types of uncertainty that we wish to account for in this situation and review previous work in this area. Then, in Section 6.1.4, we introduce a graphical model which represents the relationships between the components of the model, considering different aspects of the belief separation structure implied by the diagram, and in Section 6.1.6, we consider how we might quantify the diagram in both the fully probabilistic and Bayes linear cases. In Section 6.1.7, we consider the Bayes linear case in detail; we illustrate this case through application to a simple problem in Section 6.1.8. In Section 6.2, we go on to consider a more complex example.

6.1.1 Numerical schemes

A first-order ODE model specifies the first derivatives and initial values of a set $u(t) = \{u_1(t), \ldots, u_{n_u}(t)\}$ of functions

$$\begin{aligned} \frac{d}{dt}(u_i(t)) &= f_i(u(t), t, \xi) \\ u_i(t_0) &= u_i^{(0)} \end{aligned} \tag{6.1.1}$$

where $f(.) = \{f_1(.), \ldots, f_{n_u}(.)\}$ is a set of known functions which govern the behaviour of the solution derivatives, $u^{(0)} = \{u_1^{(0)}, \ldots, u_{n_u}^{(0)}\}$ is a set of initial values and $\xi = \{\xi_1, \ldots, \xi_{n_\xi}\}$ is a set of parameters which control the derivatives through f. Taken together, this system of differential equations and set of initial conditions is enough to determine the solution trajectory for all values of t; the solution to the problem can be written in integral form as

$$u_i(t) = u_i^{(0)} + \int_{t_0}^{t} f_i(u(s), s, \xi)\, ds \tag{6.1.2}$$

though we can only actually identify the function u by performing this integral in a very limited number of special cases.

If we cannot solve the system (6.1.1) analytically, we must choose a numerical scheme to use to approximate the solution. All of the numerical schemes which we will consider have the following common elements; we choose a set of monotonically increasing knots $\{t_0 < t_1 < t_2 < \ldots < t_{n_t}\}$, and we recursively determine an

approximate solution $\hat{u}$ at these knots as

$$\hat{u}_i\left(t_{k+1}\right) = \phi_i\left(\hat{u}\left(t_k\right), t_k, t_{k+1}, \xi\right)$$

beginning from the initial point $\hat{u}\left(t_0\right) = u^{(0)}$. Our numerical scheme is the form we choose for the functions ϕ; various different possibilities exist, and we consider some of the most common below.

Euler scheme The simplest, and perhaps the most common, numerical scheme for ODE models is the Euler scheme; we evolve by approximating the derivative as constant over the duration of the time step. The evolution function is as follows

$$\phi_i\left(\hat{u}\left(t_k\right), t_k, t_{k+1}, \xi\right) = \hat{u}_i\left(t_k\right) + h_k f_i\left(\hat{u}\left(t_k\right), t_k, \xi\right)$$

where $h_k = t_{k+1} - t_k$. Euler schemes are extremely simple to implement, but approximate only the first-order behaviour of the solution; for sufficiently small time-steps, though, they can still provide an adequate approximation to the solution.

Runge-Kutta scheme Runge-Kutta schemes are designed to capture higher-order derivative behaviour over the time-step (see, for example, Schober et al. [2014] for a brief introduction or Iserles [2008] for a more comprehensive treatment). A Runge-Kutta method of stage s uses s evaluations $w_p = \{w_{1p}, \ldots, w_{n_u p}\}$, $p = 1, \ldots, s$ of the derivative function collected recursively at points $v_p = \{v_{1p}, \ldots, v_{n_u p}\}$ as

$$w_{ip} = f_i\left(v_p, t_k + c_i h_k, \xi\right) \qquad v_{ip} = \hat{u}_i\left(t_k\right) + h_k \sum_{q=1}^{i-1} a_{pq} w_{iq} \ .$$

The solution approximation at time $t_{k+1} = t_k + h_k$ is then obtained by re-combining these steps as

$$\hat{u}_i\left(t_{k+1}\right) = \hat{u}_i\left(t_k\right) + h_k \sum_{p=1}^{s} b_p w_{ip} \ .$$

The coefficients $\{b_p\}$, $\{c_p\}$ and $\{a_{pq}\}$ are generally chosen so that the expressions for the numerical scheme and the expressions for the Taylor expansion of the solution coincide up to a particular order; for further discussion of this, see the book by Iserles. These schemes are designed to give greater accuracy than the Euler scheme; this allows us to have greater confidence in the numerical solution over longer time-steps.

The trade-off for this greater accuracy is the increase in the computational effort required for each time-step, since we need to make multiple function evaluations for each. This raises the question of whether such a more complex scheme is warranted for a particular application; is there a point at which its accuracy is matched and its performance is outstripped by simply running an Euler scheme on a finer resolution? To find out, we must set this problem within a statistical framework.

6.1.2 Numerical discrepancy

The use of a numerical scheme to approximate the solution to the system (6.1.1) produces an approximate solution surface which is discrepant from the true solution by some unknown amount. While this discrepancy is unknown, we will generally have beliefs about its likely magnitude across any given time step (possibly based on analysis of the behaviour of the scheme), which we can incorporate into our overall uncertainty specification. Additionally, its behaviour is highly systematic, as a result of the structure of the calculation which generated it; the numerical discrepancy at any time step is a smooth function of its inputs $\{\hat{u}\left(t_k\right), t_k, t_{k+1}, \xi\}$, which is generally strongly correlated across time steps. Failure to account for the effect of the numerical discrepancy will result in a cumulative error across time-steps, owing to the recursive nature of the numerical solution procedure.

We define the numerical discrepancy at time t_{k+1} as the difference between the true solution at this point and the approximation generated under our scheme, assuming that we numerically evolved from the true solution $u\left(t_k\right)$ at time t_k, so that

$$u_i\left(t_{k+1}, \xi\right) = \hat{u}_i\left(u\left(t_k\right), t_k, t_{k+1}, \xi\right) + \eta_i\left(u\left(t_k\right), t_k, t_{k+1}, \xi\right) \ . \tag{6.1.3}$$

That is: if we start from the point $\left(t_k, u\left(t_k\right)\right)$ on the true solution trajectory, and evolve numerically to time t_{k+1}, then our approximation $\hat{u}$ will differ from the true solution at t_{k+1} by an amount η, which is a function of the same inputs as the numerical scheme.

We examine the behaviour of the numerical discrepancy generated under the Euler scheme by considering the Taylor expansion of the solution to (6.1.1) around the k^{th}

time knot

$$u_i(t,\xi) = u_i(t_k,\xi) + (t-t_k)\frac{du}{dt}(t_k,\xi) + \frac{1}{2!}(t-t_k)^2\frac{d^2u}{dt^2}(t_k,\xi) + \frac{1}{3!}(t-t_k)^3\frac{d^3u}{dt^3}(t_k,\xi) + \dots .$$

Using the fact that $\frac{du}{dt}(t_k) = f_i(u(t), t, \xi)$, the difference between the true solution at t_{k+1} and the approximation under the Euler scheme is

$$\begin{aligned}\eta_i(u(t_k), t_k, t_{k+1}, \xi) &= u_i(t_{k+1},\xi) - \hat{u}_i(u(t_k), t_k, t_{k+1}, \xi) \\ &= \frac{1}{2!}(t_{k+1}-t_k)^2\frac{d^2u}{dt^2}(t_k,\xi) + \frac{1}{3!}(t_{k+1}-t_k)^3\frac{d^3u}{dt^3}(t_k,\xi) + \dots \\ &= \frac{1}{2!}h_k^2\frac{d^2u}{dt^2}(t_k,\xi) + \frac{1}{3!}h_k^3\frac{d^3u}{dt^3}(t_k,\xi) + \dots .\end{aligned}$$

Since the Euler scheme can also be viewed as a Taylor expansion truncated at first order, the discrepancy between it and the solution is, by definition, the (potentially infinite) series of higher-order terms. While it is not possible to evaluate the discrepancy exactly in this way, for small time-steps, it can be approximated (to a required degree of accuracy) using some of the remaining low-order terms; this can be used as the basis for a statistical representation of the discrepancy. Similar analyses can be carried out for other types of numerical scheme, to determine the leading-order behaviour of the numerical discrepancy.

6.1.3 Bayesian analysis for numerical schemes

There has been a great deal of interest recently in the development of Bayesian methods to track uncertainties through the numerical modelling problem outlined in Section 6.1.1. For a given numerical scheme and a fixed setting of the inputs ξ, it is simple to generate a numerical approximation to the solution trajectory for a particular ODE system. However, as in the general Bayesian uncertainty analysis problem outlined in Section 2.4.2, we will generally be uncertain about which settings of ξ will generate solution surfaces which give an adequate representation of the system that we are exploring, and about the discrepancy between the numerical solution generated at this setting and the system that our model represents. In a general Bayesian uncertainty analysis, we would fit a model to the deterministic numerical solution $\hat{u}$, use this model to approximate the uncertainty in the numerical

solution surface induced by uncertainty about the input parameters, and then use system data to fit a model to the discrepancy between our approximate solution and the real system. In this case though, the problem structure outlined in Sections 6.1.1 and 6.1.2 provides us with an opportunity to separate out the discrepancy arising through the numerical approximation to the solution from the discrepancy arising through the inadequacy of the model as originally specified (for example, due to physical processes neglected in our treatment).

The complete set of uncertainties that we wish to account for in our model is summarised below:

- **Initial condition uncertainty:** The starting point for the trajectory, $u^{(0)}$ is subject to uncertainty;
- **Model input uncertainty:** We are uncertain about the settings ξ^* of the ODE model parameters which give an adequate representation of the system under study;
- **Numerical discrepancy:** The use of a numerical solver introduces an unknown discrepancy between our approximation to the solution and the underlying, true solution of the original equations;
- **Model discrepancy:** Even if we did have access to the true solution to the ODEs, this true solution would not generally give a perfect description of the system that it represents, owing to, for example, physical processes neglected in our model specification.

A number of authors have previously focussed on handling some or all of the above sources of uncertainty. Graepel [2003] produces a closed-form probabilistic representation for the solution of a linear ODE by applying the differential operator to a Gaussian process (making use of the properties outlined in Section 2.3.2) and then sampling the forcing function at a number of locations. This approach achieves some success, though estimating the parameters of the covariance function in this context can be particularly challenging, since there may not be a particular value which adequately describes the properties of the solution across the whole of the

solution domain. Fuentes et al. [2003] and Guttorp and Walden [1987] propose a representation of simulator output and of system data which allows for the presence of both smoothing errors introduced through the discretization and numerical errors introduced through implementation of the simulator predictions.

Following in the footsteps of those who used Gaussian processes to produce uncertainty specifications for the results of numerical integrals (for example, Diaconis [1986], O'Hagan [1987], O'Hagan [1991]), Chkrebtii et al. [2016] adopt an approach in which a Gaussian process is fitted to the derivative equations and the solution is approximated by integrating the emulator across each time-step as in (6.1.2) (again, using the properties outlined in Section 2.3.2). Parameter uncertainties and the effect of the choice of knots on the solution are then handled through a sampling scheme. Hennig et al. [2015] give a general 'call to arms' for probabilistic numerical methods, which handle the numerical uncertainties induced thorough the approximation of intractable calculations using a probabilistic framework.

Alternatively, Conrad et al. [2017] solve ODEs probabilistically by introducing stochastic representations of the numerical discrepancy into traditional numerical solvers (e.g. Euler scheme, Runge-Kutta scheme, as in Section 6.1.1) and sampling the solution trajectory. These probabilistic solvers are shown to converge to the true solution in the limit as the time-step goes to zero, and are shown through numerical experiments to produce a more faithful description of our uncertainty about the solution than traditional numerical solvers which ignore the accumulation of numerical error across time-steps. The model that we present in the following Section (6.1.4) is most closely related to this last framework.

6.1.4 Model specification

Owing to the large number of elements in this problem, a graph is particularly useful as a representation of the structure of our beliefs; Figure 6.1 is a DAG which qualitatively represents the belief structure implied by the discussion in 6.1.2. As in the Bayesian uncertainty analysis framework outlined in Section 2.4.2:

- We denote the value of the real system at time t_k by $y\left(t_k\right) = \{y_1\left(t_k\right), \ldots, y_{n_u}\left(t_k\right)\}$;

- We assume that $y_i(t_k) = u_i(t_k, \xi^*) + \delta_i(t_k)$, where ξ^* is the unknown 'best input' parameter setting for the model, and $\delta_i(t_k)$ is the discrepancy between the solution to the ODE model and the system at time t_k, which is assumed to be uncorrelated with the u;

- Observations on the system at time t_k are denoted by $z_k = \{z_{1k}, \ldots, z_{n_u k}\}$, with $z_{ik} = y_i(t_k) + \epsilon_{ik}$, where the $\{\epsilon_{ik}\}$ are measurement error terms which are uncorrelated across time-steps.

Notably, the graph in Figure 6.1 implies that:

- $u(t_k, \xi^*)$ is separated from the solutions $u(t_l, \xi^*)$ at all earlier times $t_l < t_k$ by the pair $\{\hat{u}(u(t_{k-1}), \xi^*), \eta(u(t_{k-1}), \xi^*)\}$ (this property is implied by the representation (6.1.3));

- The system value $y(t_k)$ is separated from all other components of the solution trajectory by the pair $\{u(t_k, \xi^*), \delta(t_k)\}$;

- The parent set of the numerical discrepancy $\eta(u(t_{k-1}), \xi^*)$ at time t_k contains the numerical discrepancies at all earlier times and the best input parameter setting;

- The parent set of the discrepancy $\delta(t_k)$ contains the discrepancy terms at all earlier times.

The DAG representation of our belief structure is useful for making our initial prior specification for this problem; as discussed in Section 2.1.3, in order to fully characterise a directed graphical model, we must only prior specify expectations and variances for each node, and prior covariances between each node and its parents. A procedure for generating a prior specification for the DAG is discussed in Section 6.1.5. It is more difficult to use the DAG to adjust beliefs in the light of observations on the system; for this purpose, we construct a junction tree which corresponds to the DAG.

Junction tree A general procedure for constructing a junction tree from a directed graphical model is outlined in Section 2.1.3. A junction tree is an undirected

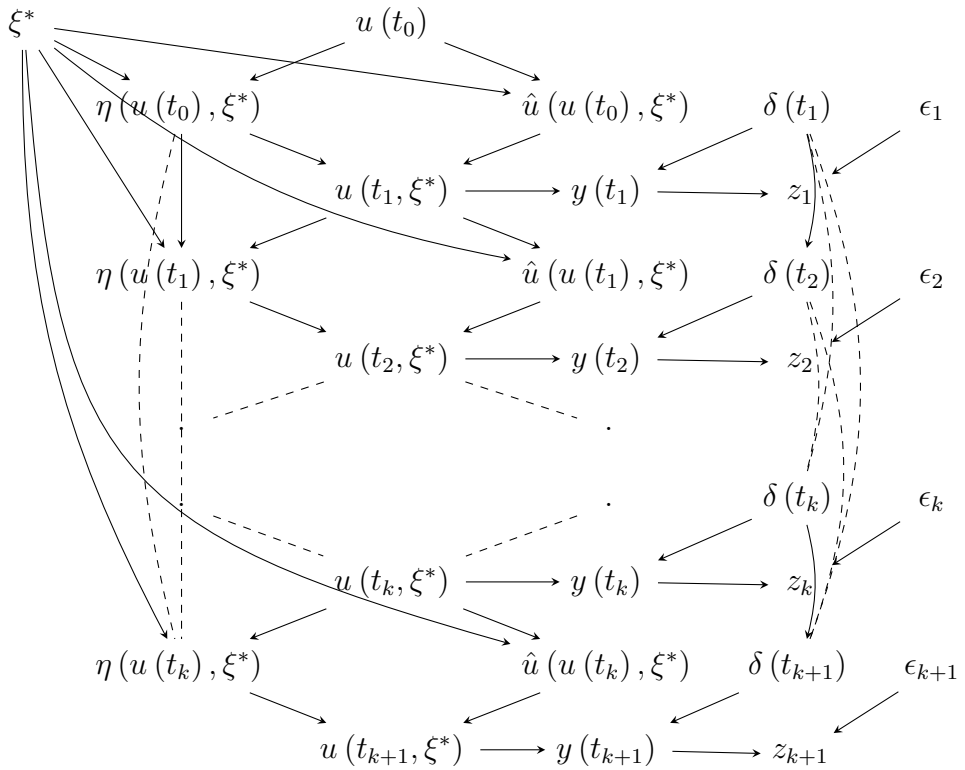


Figure 6.1: DAG representing the structure of the model for the ODE solution u and its relationship to the system y.

graphical model which is formed using the cliques of the triangulated moral graph; the fact that it has a tree structure allows information to be propagated efficiently around the graph, allowing for computationally efficient belief updates given observed data. Figure 6.2 shows a triangulated moral graph derived from a simplified version of the DAG 6.1. Note that we have dropped all existing edges between the discrepancy elements $\{\delta(t_k)\}$; we will re-visit this choice in a few paragraphs' time. To obtain this moral graph, we have performed the following steps:

- All edges from the initial DAG 6.1 are retained, with their directionality removed- these edges are shown in green;
- An edge is introduced between all unconnected pairs of nodes with a common child in the original DAG:
  - the numerical discrepancy $\eta(u(t_{k-1}), \xi^*)$ at time-step k is joined to the corresponding numerical solution $\hat{u}(u(t_{k-1}), \xi^*)$;
  - the solution $u(t_k, \xi^*)$ at time t_k is joined to the numerical discrepancies $\eta(u(t_{k-2}), \xi^*), \ldots, \eta(u(t_1), \xi^*)$ at all earlier times;
  - the solution $u(t_k, \xi^*)$ at time t_k is joined to the parameters ξ^*;
  - the solution $u(t_k, \xi^*)$ at time t_k is joined to the corresponding discrepancy $\delta(t_k)$;
  - the measurement error ϵ_k at time t_k is joined to the corresponding system value $y(t_k)$.

  Edges introduced in this way are shown in red;
- No additional edges are required in this instance in order to triangulate the graph.

From this graph, we identify the cliques. We identify the following set of cliques for each time-step t_k:

- $Q_1(t_k) = \{u(t_k, \xi^*), \eta(u(t_k), \xi^*), \ldots, \eta(u(t_0), \xi^*), \xi^*\}$ for $k = 1, \ldots n_t$;
- $Q_2(t_k) = \{u(t_k, \xi^*), \eta(u(t_{k-1}), \xi^*), \hat{u}(u(t_{k-1}), \xi^*), \xi^*\}$ for $k = 1, \ldots n_t$;

- $\mathrm{Q}_3(t_k) = \{\eta(u(t_{k-1}), \xi^*), \hat{u}(u(t_{k-1}), \xi^*), u(t_{k-1}, \xi^*), \xi^*\}$ for $k = 1, \ldots n_t$;
- $\mathrm{Q}_4(t_k) = \{u(t_k, \xi^*), y(t_k), \delta(t_k)\}$ for $k = 1, \ldots, n_t$;
- $\mathrm{Q}_5(t_k) = \{y(t_k), z_k, \epsilon_k\}$ for $k = 1, \ldots, n_t$.

Figure 6.3 shows the junction tree which is formed from these cliques. As outlined in Section 2.1.3, we can use this graph as a tool for adjusting beliefs about all model components given observation of a particular z_k; to obtain adjusted beliefs for the model components, we simply adjust by each data element in turn, using the adjusted moments after an individual update as the prior moments for the next.

Adjusting using this junction tree eliminates the need for us to compute and invert a covariance matrix the size of the full set of model components in order to perform the adjustment, and so helps us to make progress where it would be computationally prohibitive to do so. However, due to the complete graph formed by the $\{\eta(u(t_{k-1}), \xi^*)\}$, it may still be time-consuming to repeatedly propagate adjustments through the junction tree. One strategy for reducing this computational complexity is to drop some of the edges between these components; for example, we might choose to link $\eta(u(t_{k-1}), \xi^*)$ to only $\{u(t_{k-1}, \xi^*), \ldots, u(t_{k-q-1}, \xi^*)\}$ for some $q \geq 0$, in order to preserve some of the structure in the numerical discrepancy model, while lessening the computational burden. This is the strategy adopted for the example presented in Section 6.2.

The decision to drop the edges present in the DAG 6.1 between the discrepancy components $\{\delta(t_k)\}$ was made in order to make the identification of the junction tree for the ODE solution model simple. Restoring the complete graph formed by these components in the original directed graph would mean that we would also be required to introduce a large number of additional edges in order to triangulate the resulting graph. Identifying a general way to triangulate the graph with a full discrepancy covariance structure will be the object of future research in this area.

6.1.5 Quantifying the diagram

We can now use the graph 6.1 to specify a full joint prior distribution across the components of the model. As with the other modelling problems discussed in previ-

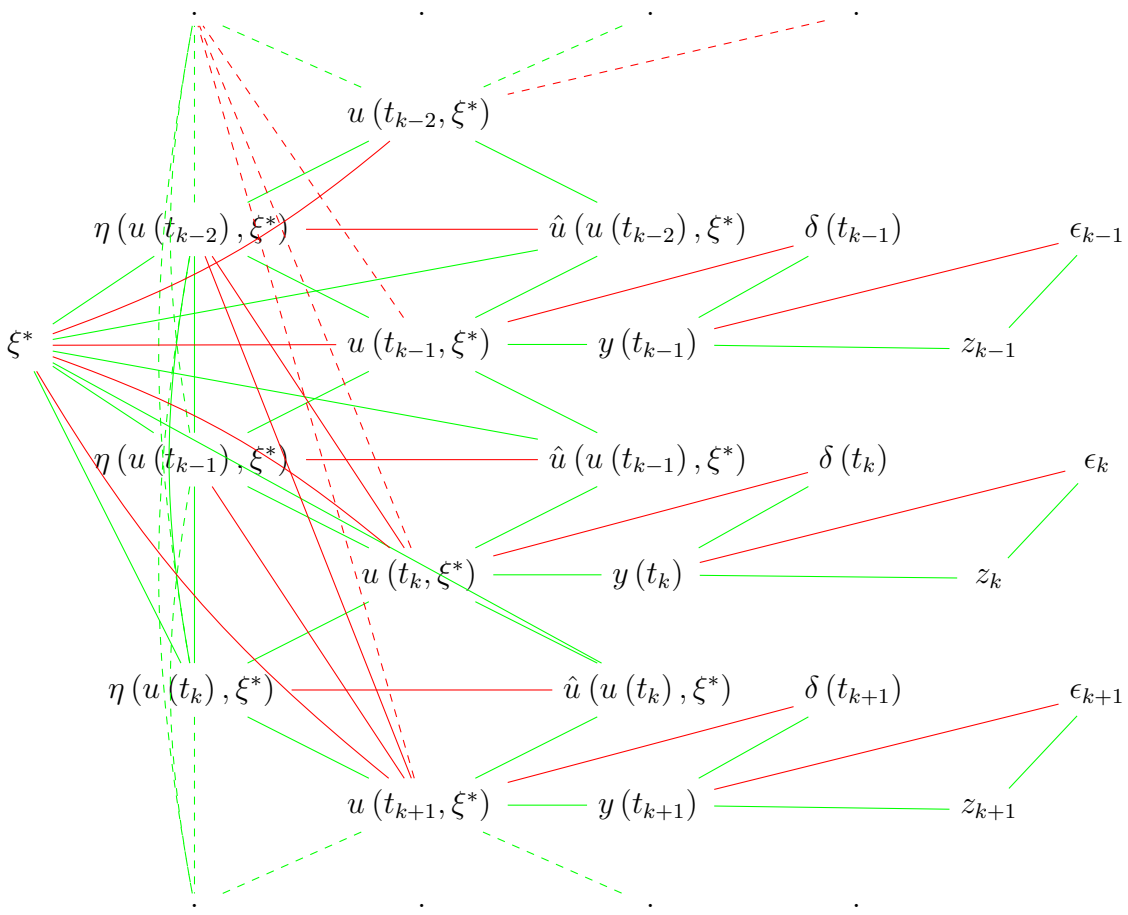


Figure 6.2: Triangulated moral graph corresponding to a simplified version of the DAG 6.1. Edges are coloured according to how they are introduced: green edges are present in the original DAG, and red edges are introduced through moralization ('marrying the parents') in the DAG.

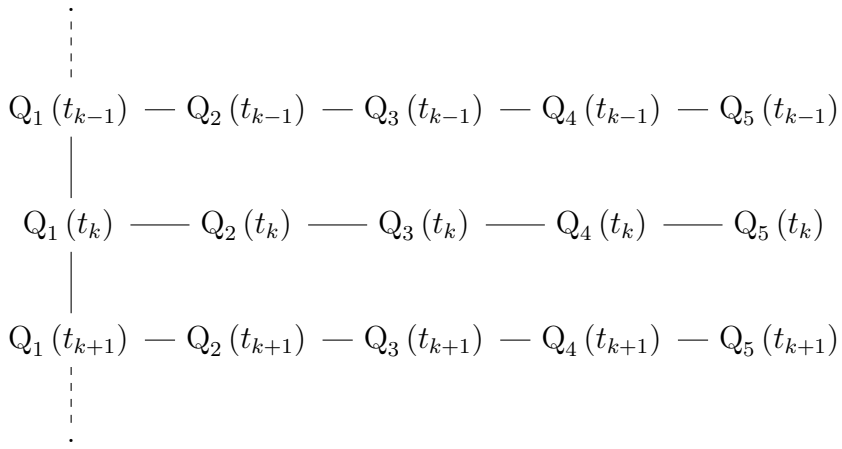


Figure 6.3: Junction tree formed from the cliques of the triangulated moral graph 6.2.

ous chapters, there are two options available to us when it comes to quantifying the diagram; we can either make a fully probabilistic specification for all components, or we can make a second-order specification for all nodes and all edges between nodes. We consider both of these options below.

Probabilistic specification To make a full joint probability specification for all of the components of 6.1, we must specify a distribution $p\left(v|\mathrm{Pa}\left(v\right),\dots\right)$ for each component conditional on its parent nodes (and any other (hyper)parameters which are not displayed on the graph); the graph and the component distributional specifications then implicitly define a joint prior probability distribution over all components. Where we have specific beliefs about the distributions of individual components, then we should adopt this probabilistic approach.

Where we do choose to make a fully probabilistic specification, we will not, in general, be able to algebraically derive conditional or marginal relationships between non-adjacent components, or marginal distributions for individual nodes. Indeed, we could only find all these distributions algebraically in the situation where we make a conditional linear Gaussian specification for all elements, and in which the solver and numerical discrepancy input relationships are linear; clearly an unrealistically simple specification for most problems of interest. We will therefore need to rely on numerical sampling schemes in order to characterise the marginal prior

distribution of system values $\{y(t_k)\}$, or the posterior distribution of these system values given observation of the data z_k at a handful of time-steps.

For such an analysis, the DAG 6.1 can also be used to design an efficient sampling scheme; the fact that we have specified the full joint distribution through its conditional distributions means that it is simple to implement a Gibbs procedure, in which we cycle through the components, updating the current state of each conditional on the current states of all others. Where non-Gaussian distributions are chosen, we must implement a Metropolis-Hastings step for each component update; if the use of these steps results in slow exploration of the distribution, then we have the option of using gradient-based proposal mechanisms which exploit local knowledge of the shape of the conditional distributions in order to generate better proposals (at an increased computational cost; see Girolami and Calderhead [2011] for further discussion of these methods).

Second-order analysis In the situation where we do not have strong prior beliefs about the distributions of the individual components, it may be more appropriate from both a foundational and practical viewpoint to make a joint second-order prior specification for all components and carry out a Bayes linear analysis. As discussed in Section 2.1.2, in specifying only first- and second-order moments, we avoid making distributional assumptions that we do not necessarily believe; also, as will be discussed further in Section 6.1.6, we reduce the computational complexity of the calculations which we must carry out in order to learn about the components of the model from data.

As in the probabilistic case, imposition of a particular graphical structure (Figure 6.1) for our model reduces the complexity of the specification needed to generate a full joint prior. If we specify expectations and variances for each of the nodes, and covariances corresponding to the edges in the DAG, then the methodology outlined in Section 2.1.3 allows us to compute the covariance between any pair of nodes in the graph, and therefore provides us with access to the corresponding full joint second-order prior specification. For computational reasons, it may sometimes be better to specify covariances corresponding to edges in the moral graph directly,

rather than initially characterising the DAG; this point will be discussed further in Section 6.1.6.

Our second-order specification will generally be insufficient for propagating uncertainty about the solution at one time point through the numerical solver for the next. As with the uncertainty propagation calculations for an emulator (Section 2.4.2), we will generally need to specify expectations and (co)variances for higher-order polynomial terms; typically, we will do this through choice of a particular distribution, usually a Gaussian characterised by our expectation and covariance specification.

6.1.6 Bayes linear prior specification

We now focus on the Bayes linear analysis of a general problem of the form represented by the graph 6.1. We make a second-order prior specification for the initial state and the best input parameters, we choose a numerical scheme along with a numerical integration technique, we specify a model for the structure of the corresponding numerical discrepancy term, and then we numerically compute the elements of the full prior structure implied by these components and the graph 6.1. In Section 6.1.7, we then discuss the use of the graph to compute the adjusted moments upon observation of some of the data components.

Limited prior specification The very first thing that we need to do is specify our prior beliefs about the initial solution state $u^{(0)}$, and about the best input setting ξ^*. We specify first- and second-order prior moments $\mathrm{E}\left[u^{(0)}\right]$, $\mathrm{Var}\left[u^{(0)}\right]$, $\mathrm{E}\left[\xi^*\right]$ and $\mathrm{Var}\left[\xi^*\right]$. This specification, when coupled with a particular choice of numerical scheme and a model for the resulting numerical discrepancy, will determine the characteristics of the solution for all future time-steps.

Numerical solver We must choose a solver, with which we will generate our numerical solution estimates at each time step. In practice, this will be either an Euler solver (possibly augmented with higher-order terms from the Taylor expansion of the solution) or a Runge-Kutta solver; see Section 6.1.1. Our choice of solver

function determines our solution approximations as

$$\hat{u}_i\left(u\left(t_k\right), t_k, t_{k+1}, \xi\right) = \phi_i\left(u\left(t_k\right), t_k, t_{k+1}, \xi\right) \ .$$

In order to be able to assess the expectation and variance of $\hat{u}$, and its covariance with adjacent nodes, we must be able to propagate uncertainty on $\{u\left(t_k\right), \xi^*\}$ through our solver function ϕ. In order to do this, we will generally need to specify the expectations and covariances of higher order terms in the expression for ϕ; we will generally do this through the specification of a distribution $p\left(u\left(t_k\right), \xi\right)$.

If the expression for ϕ contains terms for which expectations and covariances cannot be evaluated directly (e.g. sines and cosines, exponentials), then we can assess $\mathrm{E}\left[\phi\right]$, $\mathrm{Var}\left[\phi\right]$ empirically through sampling (for particular distributional assumptions). Alternatively, we can approximate these moments by emulating ϕ and propagating our uncertainty through the emulator, as described in Section 2.4.2. Emulation is also a useful strategy in the situation where f, and therefore ϕ, are complex functions which are slow to evaluate.

Numerical discrepancy A particular choice for $\phi\left(.\right)$ implicitly defines a numerical discrepancy function which, as discussed in Section 6.1.2, we cannot evaluate directly. While evaluating this function is not possible, we can perform a prior analysis which will enable us to model its main effects, and will therefore improve the accuracy of our model for the solution. We generate numerical discrepancy data by running the solver at a finer resolution for a handful of time intervals, producing a more accurate estimate of the solution and a noisy 'observation' of the corresponding discrepancy; this data is used to fit a simple model, which is then used as an input to the remainder of our calculations.

We begin by specifying a simple uncorrelated model for the discrepancy, $\eta_0(h)$, with mean zero and standard deviation proportional to the time-step ($\mathrm{Var}\left[\eta_0(h)\right] = (\sigma_\eta h)^2$), so that for evolution over any interval $[t_k, t_{k+1}]$

$$u_i\left(t_{k+1}, \xi\right) = \hat{u}_i\left(u\left(t_k\right), t_k, t_{k+1}, \xi\right) + \eta_i^0(h_k) \ .$$

Over the interval $[t_k, t_{k+1}]$, we now define a refined grid of n_k^* knots, $\{t_{k0}^*, \ldots, t_{kn^*}^*\}$ (where $t_{k0}^* = t_k$ and $t_{kn^*}^* = t_{k+1}$), and the solver is run from t_k to t_{k+1} using this

refined grid, sampling the $\eta^0(h^*_{kp})$ independently from a Gaussian distribution for each $h^*_{kp} = t^*_{kp} - t^*_{k(p-1)}$. We denote the resulting 'fine grid' sample of the solution at t_{k+1} by $u^*(u(t_k), t_k, t_{k+1}, \xi)$; $\mathrm{E}[u^*]$ and $\mathrm{Var}[u^*]$ are characterised by repeating this sampling procedure a number of times using the same inputs. Based on this analysis, our second-order beliefs about the numerical discrepancy incurred by running the coarse solver over these time intervals are

$$\begin{aligned}\mathrm{E}[\eta_i(u(t_k), t_k, t_{k+1}, \xi)] &= \mathrm{E}[u_i^*(u(t_k), t_k, t_{k+1}, \xi)] - \hat{u}_i(u(t_k), t_k, t_{k+1}, \xi) \\ \mathrm{Var}[\eta_i(u(t_k), t_k, t_{k+1}, \xi)] &= \mathrm{Var}[u_i^*(u(t_k), t_k, t_{k+1}, \xi)] \ .\end{aligned}$$

We repeat this procedure for a number of different input settings (initial and final times, initial conditions and parameter settings); note that the time knots used do not need to correspond to those that we will eventually use for our trajectory analysis, though running the deterministic numerical scheme forward from a sample of the initial condition $u^{(0)}$ is a useful way to approximately constrict the data that we generate to the set of initial points $\{t_k, u(t_k)\}$ that we may actually reach in our final analysis.

We now use this data to fit a model to the discrepancy; we assume this to have the common 'regression plus residual' form

$$\eta_i(u(t_k), t_k, t_{k+1}, \xi) = \sum_{p=1}^{n_{g_\eta}} \beta_{ip} g_p(u(t_k), t_k, t_{k+1}, \xi) + r_i(u(t_k), t_k, t_{k+1}, \xi) \ . \quad (6.1.4)$$

As when we were fitting emulators to risk functions in chapters 3, 4 and 5, we may exploit our knowledge of the likely behaviour of the underlying function when selecting basis functions. In this instance, choosing the elements of g to be terms from the neglected higher order components of the Taylor expansion of the solution at t_k will generally produce a good model for discrepancy behaviour.

Full prior specification Once we have specified each of the above components, we may use the graph to generate a full joint prior specification for each of the components. To do this, we work through the graph in order, using each node's relationship with its parents to derive a node expectation and variance specification and a covariance specification corresponding to the graph's edges. Beginning from

our initial specification $\mathrm{E}\left[\xi^*\right], \mathrm{Var}\left[\xi^*\right]$ and $\mathrm{E}\left[u^{(0)}\right], \mathrm{Var}\left[u^{(0)}\right]$, we cycle through the time knots t_k for $k = 1, \ldots, n_t$ computing:

- moments $\mathrm{E}\left[\hat{u}\left(t_k\right)\right]$ and $\mathrm{Var}\left[\hat{u}\left(t_k\right)\right]$ of the numerical approximation, and covariances $\mathrm{Cov}\left[\hat{u}\left(t_k\right), u\left(t_{k-1}\right)\right]$ and $\mathrm{Cov}\left[\hat{u}\left(t_k\right), \xi^*\right]$ of this component with its parents;

- moments $\mathrm{E}\left[\eta\left(t_k\right)\right]$ and $\mathrm{Var}\left[\eta\left(t_k\right)\right]$ of the numerical discrepancy, covariances $\mathrm{Cov}\left[\eta\left(t_k\right), u\left(t_{k-1}\right)\right]$, $\mathrm{Cov}\left[\eta\left(t_k\right), \xi^*\right]$ with the discrepancy inputs, and covariances $\mathrm{Cov}\left[\eta\left(t_k\right), \eta\left(t_l\right)\right]$ for $l = (k-1), \ldots, 1$ with previous discrepancy terms;

- moments $\mathrm{E}\left[u\left(t_k\right)\right]$ and $\mathrm{Var}\left[u\left(t_k\right)\right]$ of the solution, and covariances $\mathrm{Cov}\left[u\left(t_k\right), \hat{u}\left(t_k\right)\right]$ and $\mathrm{Cov}\left[u\left(t_k\right), \eta\left(t_k\right)\right]$ of this component with its parents.

We consider each of these components in turn. In the case of the numerical solution $\hat{u}$, we must propagate input uncertainty through the solver

$$\begin{aligned}
\mathrm{E}\left[\hat{u}_i\left(t_k\right)\right] &= \mathrm{E}\left[\mathrm{E}\left[\phi_i\left(\theta\right)\right]\right] \\
\mathrm{Cov}\left[\hat{u}_i\left(t_k\right), \hat{u}_j\left(t_k\right)\right] &= \mathrm{E}\left[\mathrm{Cov}\left[\phi_i\left(\theta\right), \phi_j\left(\theta\right)\right]\right] + \mathrm{Cov}\left[\mathrm{E}\left[\phi_i\left(\theta\right)\right], \mathrm{E}\left[\phi_j\left(\theta\right)\right]\right]
\end{aligned}$$

where in this context $\theta = \left\{u\left(t_{k-1}\right), t_{k-1}, t_k, \xi^*\right\}$, and the outer expectations and covariances are taken with respect to $\left\{u\left(t_{k-1}\right), \xi^*\right\}$. The covariance of this numerical approximation with its inputs is then

$$\begin{aligned}
\mathrm{Cov}\left[\hat{u}_i\left(t_k\right), u_j\left(t_{k-1}\right)\right] &= \mathrm{E}\left[\mathrm{E}\left[\phi_i\left(\theta\right)\right] u_j\left(t_{k-1}\right)\right] - \mathrm{E}\left[\hat{u}_i\left(t_k\right)\right] \mathrm{E}\left[u_j\left(t_{k-1}\right)\right] \\
\mathrm{Cov}\left[\hat{u}_i\left(t_k\right), \xi_j^*\right] &= \mathrm{E}\left[\mathrm{E}\left[\phi_i\left(\theta\right)\right] \xi_j^*\right] - \mathrm{E}\left[\hat{u}_i\left(t_k\right)\right] \xi_j^* \ .
\end{aligned}$$

To compute expectations of any non-linear terms in the above expressions, we must make a specification for the higher-order moments of the components of these expressions; we do this by characterising a probability distribution (generally a Gaussian) $p\left(u\left(t_{k-1}\right), \xi^*\right)$ using the moments of these components and either approximating the expectation by sampling or integrating directly.

Propagating uncertainty through the numerical discrepancy component is, in general, more difficult, owing to the prior correlation imposed between this and all other discrepancy components. As in the case of the numerical approximation, we

can write the expectation and covariance of $\eta_i(t_k)$ for given settings of $\mathrm{Pa}(\eta_i(t_k))$, and we can compute expectations and covariances of these

$$\begin{aligned}
\mathrm{E}[\eta_i(t_k)] &= \mathrm{E}\left[\mathrm{E}_{\eta\left(t_{[k-1]}\right)}[\eta_i(\theta)]\right] \\
\mathrm{Cov}[\eta_i(t_k), \eta_j(t_k)] &= \mathrm{E}\left[\mathrm{Cov}_{\eta\left(t_{[k-1]}\right)}[\eta_i(\theta), \eta_j(\theta)]\right] \\
&\quad + \mathrm{Cov}\left[\mathrm{E}_{\eta\left(t_{[k-1]}\right)}[\eta_i(\theta)], \mathrm{E}_{\eta\left(t_{[k-1]}\right)}[\eta_j(\theta)]\right]
\end{aligned}$$

where we use $\eta\left(t_{[k-1]}\right) = \{\eta(t_1), \ldots, \eta(t_{k-1})\}$ to denote the set of discrepancy components at previous times, and in this instance, the outer expectations are taken with respect to $\{\eta\left(t_{[k-1]}\right), u(t_{k-1}), \xi^*\}$. Again, we specify the required higher-order moments by choosing a probability distribution $p\left(\eta\left(t_{[k-1]}\right), u(t_{k-1}), \xi^*\right)$.
Finally, we combine these elements to compute the moment specification for $u(t_k)$; we simply have

$$\begin{aligned}
\mathrm{E}[u_i(t_k)] &= \mathrm{E}[\hat{u}_i(t_k)] + \mathrm{E}[\eta_i(t_k)] \\
\mathrm{Cov}[u_i(t_k), u_j(t_k)] &= \mathrm{Cov}[\hat{u}_i(t_k), \hat{u}_j(t_k)] + \mathrm{Cov}[\eta_i(t_k), \eta_j(t_k)] \\
&\quad + \mathrm{Cov}[\hat{u}_i(t_k), \eta_j(t_k)] + \mathrm{Cov}[\eta_i(t_k), \hat{u}_j(t_k)]
\end{aligned}$$

where the final two covariances can be computed as, for example

$$\mathrm{Cov}[\hat{u}_i(t_k), \eta_j(t_k)] = \mathrm{E}\left[\mathrm{E}[\phi_i(\theta)]\,\mathrm{E}_{\eta\left(t_{[k-1]}\right)}[\eta_j(\theta)]\right] - \mathrm{E}[\hat{u}_i(t_k)]\,\mathrm{E}[\eta_j(t_k)]$$

where the outer expectation is taken with respect to $\{\eta\left(t_{[k-1]}\right), u(t_{k-1}), \xi^*\}$.

Relationship to the system Having computed the set of moments discussed above, relating the model for the ODE solution to the system that it represents is relatively simple. First, the solution is related to the system value by simply adding on the discrepancy component, which is assumed to be uncorrelated with the solution, so

$$\begin{aligned}
\mathrm{E}[y_i(t_k)] &= \mathrm{E}[u_i(t_k)] + \mathrm{E}[\delta_i(t_k)] \\
\mathrm{Cov}[y_i(t_k), y_j(t_k)] &= \mathrm{Cov}[u_i(t_k), u_j(t_k)] + \mathrm{Cov}[\delta_i(t_k), \delta_j(t_k)]
\end{aligned}$$

and the covariances of the system value with its parents are

$$\begin{aligned}
\mathrm{Cov}[y_i(t_k), u_j(t_k)] &= \mathrm{Cov}[u_i(t_k), u_j(t_k)] \\
\mathrm{Cov}[y_i(t_k), \delta_j(t_k)] &= \mathrm{Cov}[\delta_i(t_k), \delta_j(t_k)] \ .
\end{aligned}$$

Lastly, we compute the moments of the data by including the contribution of the (zero mean, uncorrelated) measurement error term

$$\begin{aligned} \mathrm{E}\left[z_{ik}\right] &= \mathrm{E}\left[y_i\left(t_k\right)\right] \\ \mathrm{Cov}\left[z_{ik}, z_{jk}\right] &= \mathrm{Cov}\left[y_i\left(t_k\right), y_j\left(t_k\right)\right] + \mathrm{Cov}\left[\epsilon_{ik}, \epsilon_{jk}\right] \end{aligned}$$

where the covariances corresponding to the edges in the graph are

$$\begin{aligned} \mathrm{Cov}\left[z_{ik}, y_i\left(t_k\right)\right] &= \mathrm{Cov}\left[y_i\left(t_k\right), y_j\left(t_k\right)\right] \\ \mathrm{Cov}\left[z_{ik}, \epsilon_{jk}\right] &= \mathrm{Cov}\left[\epsilon_{ik}, \epsilon_{jk}\right] \ . \end{aligned}$$

6.1.7 Bayes linear adjustment

Once we have generated the full prior specification as outlined in Section 6.1.6, we generally wish to update the model components upon learning the values of some of the $\{z_k\}$. The graph can be a useful tool here too. The most basic approach that we can take is to directly enumerate the covariances between all pairs of components, and then update each node expectation, node variance and node pair covariance using the Bayes linear update rules (Section 2.1.2). For large problems, this approach quickly becomes impractical. For a very large number of time-steps, it may not even be possible to hold all of the components of the prior specification in memory simultaneously; in such a situation, we may be forced to delete and re-compute covariances multiple times in order to stay within our memory allocation, increasing the (already potentially large) computation time for the adjustment.

For large problems, the junction tree approach to adjustment outlined in Section 2.1.3 can be extremely useful. If we find the junction tree corresponding to our graphical model specification, then we must only compute and store in memory the covariances between pairs of nodes which lie in the same clique; this generally represents a large reduction in the memory required for a particular problem, and in the number of computations which must be performed in order to characterise the prior specification, particularly in cases where we reduce the number of links between numerical discrepancy components across time-steps.

6.1.8 Example: projectile trajectory

To illustrate the proposed modelling framework, we consider a simple ODE model, taken from Kibble and Berkshire [2004]. A projectile is launched from a point $u^{(0)} = (0, 0, 0)$ in three-dimensional space at time $t_0 = 0$ with an initial velocity of $\frac{du}{dt}(t_0) = (\dot{x}_0, 0, \dot{z}_0)$. As it travels through the air, it is subject to air resistance λ and gravity f_g. Using Newton's second laws, we can write down a system of second-order ODEs which describe the motion

$$\begin{aligned}\ddot{x}(t) &= -\gamma\dot{x}(t)\\ \ddot{y}(t) &= 0\\ \ddot{z}(t) &= -\gamma\dot{z}(t) - f_g\end{aligned}$$

where $\gamma = \lambda/m$, where m is the mass of the projectile. The absence of forces acting in the y-direction, coupled with the fact that $y_0 = 0$ and $\dot{y} = 0$ means that $y(t) = 0$ for all time. Integration of the other two equations gives the following system of first-order equations

$$\begin{aligned}\dot{x}(t) &= -\gamma x(t) + \dot{x}_0\\ \dot{z}(t) &= -\gamma z(t) - f_g t + \dot{z}_0\ .\end{aligned}$$

These two equations are both in the form (6.1.1). Because of the simplicity of this system of equations, it can be solved exactly to give

$$\begin{aligned}x(t) &= \frac{\dot{x}_0}{\gamma}\big(1 - e^{-\gamma t}\big)\\ z(t) &= \Big(\frac{\dot{z}_0}{\gamma} + \frac{f_g}{\gamma^2}\Big)\big(1 - e^{-\gamma t}\big) - \frac{f_g t}{\gamma}\ .\end{aligned} \tag{6.1.5}$$

The fact that we can evaluate the solution to the system exactly in this case provides us with the opportunity to test how well the modelling framework outlined in Section 6.1.4 works by comparing with the true solution. For the remainder of this Section, we focus only on the equation describing the motion in the z-direction; we set $u(t) = z(t)$.

Numerical scheme We choose to use an Euler scheme for our numerical approximations to the solution; we have

$$\begin{aligned}\phi\left(u, t_k, t_{k+1}, \xi\right) &= u + (t_{k+1} - t_k) f\left(u, t_k, \xi\right) \\ &= u + (t_{k+1} - t_k)\Big(- \gamma u - f_g t + \dot{z}_0\Big) \ .\end{aligned}$$

This function is extremely simple to evaluate, and so we do not choose to further approximate using an emulator; doing so would not speed up the evaluation of ϕ, or assist with the propagation of uncertainty.

Numerical discrepancy model As outlined in Section 6.1.2, the choice of the Euler scheme means that the numerical discrepancy incurred by evolving from t_k consists of the higher-order terms from the Taylor expansion of u around t_k. We therefore use this expansion as the basis for our numerical discrepancy model and we use the regression plus residual form from (6.1.4). For the basis functions, we choose

$$g\left(\theta\right) = \frac{1}{2}(t_{k+1} - t_k)^2 \frac{d^2 u}{dt^2}\left(t_k\right)$$

where the second derivative is

$$\begin{aligned}\frac{d^2 u}{dt^2}\left(\theta\right) &= -\gamma \frac{du}{dt}\left(\theta\right) - f_g \\ &= -\gamma f\left(u\left(t_k\right), t_k, \xi\right) - f_g \ .\end{aligned}$$

For the residual process, we assume that $\mathrm{E}\left[\, r\left(\theta\right)\right] = 0$, and

$$\mathrm{Cov}\left[\, r\left(\theta\right), r\left(\theta'\right)\right] = V c\left(\theta, \theta'\right)$$

where V is the marginal variance of the process, and $c\left(.,.\right)$ is a squared exponential correlation function

$$\begin{aligned}c\left(\theta, \theta'\right) = \exp\Big[- \frac{1}{2}\Big(&\lambda_t (t_k - t_k')^2 + \lambda_h (h_k - h_k')^2 \\ &+ \lambda_u (u\left(t_k\right) - u\left(t_k\right)')^2 + \lambda_\xi (\xi - \xi')^2\Big)\Big] \\ \times \ \min&\left(h_k, h_k'\right) \ .\end{aligned}$$

System relationship For the purposes of this example, we simply assume that there is no systematic discrepancy between the solution to the equation and the system, so that $\delta_i(t_k) = 0$ for all i, k. For the measurement error, we simply specify that $\text{Var}[\epsilon_{ik}] = (0.01)^2$, and we will add this level of noise to all data that we synthetically generate.

Prior specification First, we specify the grid for our solver and our prior moments for the equation parameters. We consider the trajectory of a projectile in the interval $t = [0, 10]$, and we sub-divide this trajectory into 40 random intervals by fixing $t_0 = 0$, $t_{40} = 10$ and then choosing 39 additional knots $\{t_1, \ldots, t_{39}\}$ according to a Latin hypercube. Our prior beliefs about the parameters ξ^* are specified as $\text{E}[\gamma] = 0.55$, $\text{Var}[\gamma] = (0.2)^2$, $\text{E}[\dot{z}_0] = 40$, $\text{Var}[\dot{z}_0] = (10)^2$.

Our first modelling step is to fit the model for the numerical discrepancy using data generated as described in Section 6.1.6; to do this, we use the same fit procedure as was used for the risk emulators in the examples from chapters 3, 4 and 5. All of the discrepancy data for the fit is generated by choosing a random time interval, initialising the fine solver by running the original solver to the start of this interval, equally sub-dividing the randomly chosen interval into 100 elements and then sampling the fine solution 500 times in order to empirically estimate its mean and variance. Our initial iid discrepancy variance specification for this procedure is $\text{Var}[\eta^0(h)] = (0.1h)^2$.

First, a set of 200 numerical discrepancy samples is generated and used to carry out an initial linear regression; as a result, we fix the prior moments of β to $\text{E}[\beta] = 0.918$ and $\text{Var}[\beta] = (4.43 \times 10^{-4})^2$; we then use the residuals from this regression to fix $V = (0.0286)^2$. We then generate a further set of 300 samples, and we use this to perform a joint update of the regression and residual components, as outlined in Section 2.2.2. Having carried out this update, we generate a final discrepancy data set of 100 points and use this to check the model's predictions. All of the data points that we generate lie within three standard deviations of the corresponding model predictions.

Quantifying the diagram We now combine all of the components specified above with the graph (Figure 6.1) to generate the full joint prior specification; for this small problem, we simply do this by sampling the nodes in order, assuming a Gaussian distribution for each. We sample the full trajectory of the graph 5000 times, and use the resulting samples to empirically estimate the expectations and covariances for each node, and the covariances between each pair of nodes.

Results We compute the adjusted moments of all components in the model for noise-corrupted data generated from the real solution (6.1.5) using five different real settings of the parameters; Figure 6.4 shows the prior moments for the trajectory alongside the adjusted moments for each of the cases, and Figures 6.5 and 6.6 show the corresponding prior and adjusted moments for the numerical discrepancy terms and the model parameters respectively. The adjusted moments for each case are based on three observations made at locations manually chosen to give good coverage of those parts of the trajectory for which $u(t) > 0$.

We see that, while the range of trajectories possible under our prior specification is wide, the rich covariance structure imposed through the specification of the model form 6.1 means that making only 3 observations of the trajectory in each case is enough to generate an accurate prediction for the remainder of the trajectory, and an accurate estimate for the initial velocity and air resistance parameters. The fitting of the prior model (6.1.4) to the numerical discrepancy and the use of this covariance structure in the trajectory prior specification means that we can solve the ODE on a relatively coarse grid and still obtain accurate results.

6.2 Example: coupled bell-tower model

Churches around the world are equipped with sets of bells; these are used to advertise religious services and special events to those who live in the vicinity. Bells are designed to make a loud noise which can be heard over a long distance, and they are typically hung close to the top of a tower, in order to help achieve this goal. Bells which are set up for 'English style' change ringing (also referred to as 'full circle' ringing) are attached to a frame using bearings, which allows them to pivot

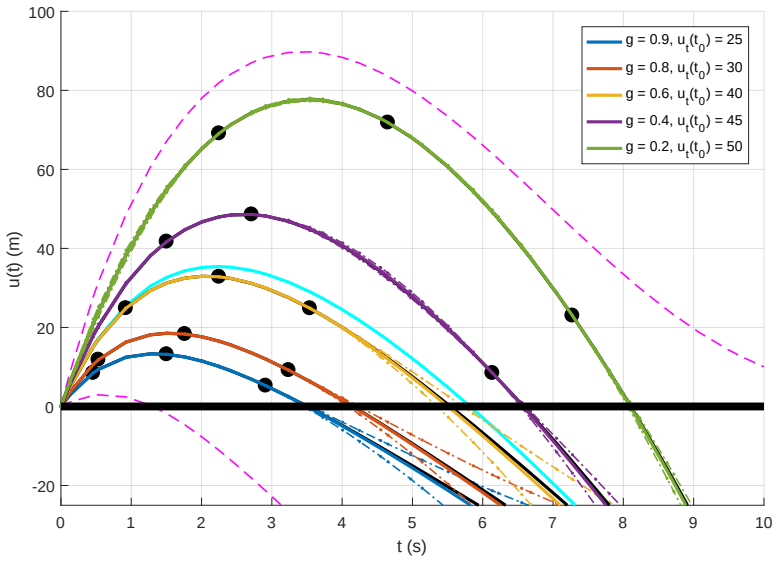


Figure 6.4: Prior moments of the trajectory $u(t)$ (with $\mathrm{E}[u(t)]$ in cyan and $\mathrm{E}[u(t)] \pm 3\mathrm{Var}[u(t)]^{1/2}$ in dashed magenta) plotted alongside separately-adjusted moments $\mathrm{E}_z[u(t)]$ (dashed) and $\mathrm{E}_z[u(t)] \pm 3\mathrm{Var}_z[u(t)]^{1/2}$ (double dashed) for data generated under 5 different parameter settings (see legend for real values corresponding to colours). 3 observations are made of each trajectory, with these observations shown as black markers and the real solutions shown as black lines for each case.

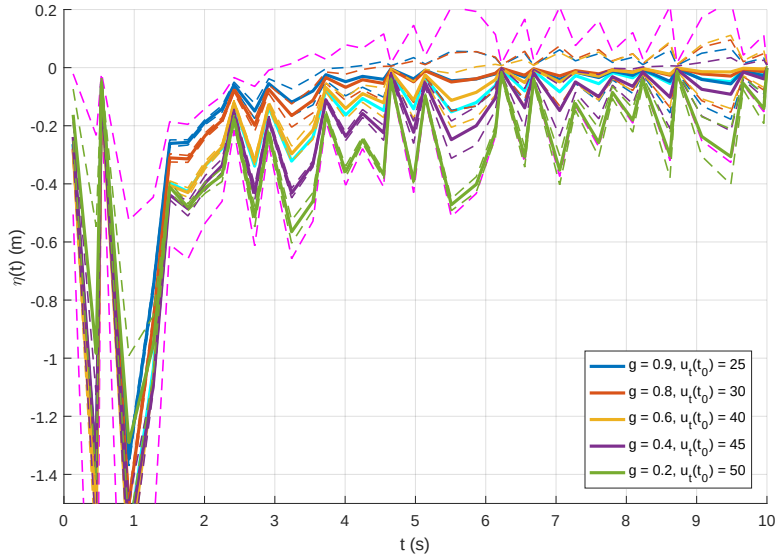


Figure 6.5: Prior ($\mathrm{E}[\eta(t)]$ in solid cyan and $\mathrm{E}[\eta(t)] \pm \mathrm{Var}[\eta(t)]^{1/2}$ in dashed magenta) and adjusted moments ($\mathrm{E}_z[\eta(t)]$ solid and $\mathrm{E}_z[\eta(t)] \pm \mathrm{Var}_z[\eta(t)]^{1/2}$ dashed, in various colours) for the numerical discrepancy components at each time step, for the same cases as in Figure 6.4.

freely; a wheel is then fixed to each bell, and it is swung through 360° by a person standing lower down the tower using a rope attached to the wheel. There are over 6000 churches in the world which have four or more bells hung for ringing in this manner, the majority of which are in the UK.

The ringing of bells in this way causes large forces to act on the towers in which they are hung. There are many towers in which the largest ringing bell has a mass in excess of 1000 kg, and the smaller bells in a set are generally sized proportional to the heaviest (for example, the second-largest bell in the ring is typically two-thirds the mass of the largest, and the third-heaviest is typically half the mass of the largest). When installing a set of bells, it is important to assess the likely effect of the forces generated through ringing on the tower which is to house them; there are numerous examples of towers where the bells may not be rung because structural analysis of the tower post-installation indicates that further ringing has the potential to cause significant damage (for example, the heavy ring of 8 at Baldersby, North Yorkshire). In less extreme cases, the ringing-induced tower motion can have an effect on the

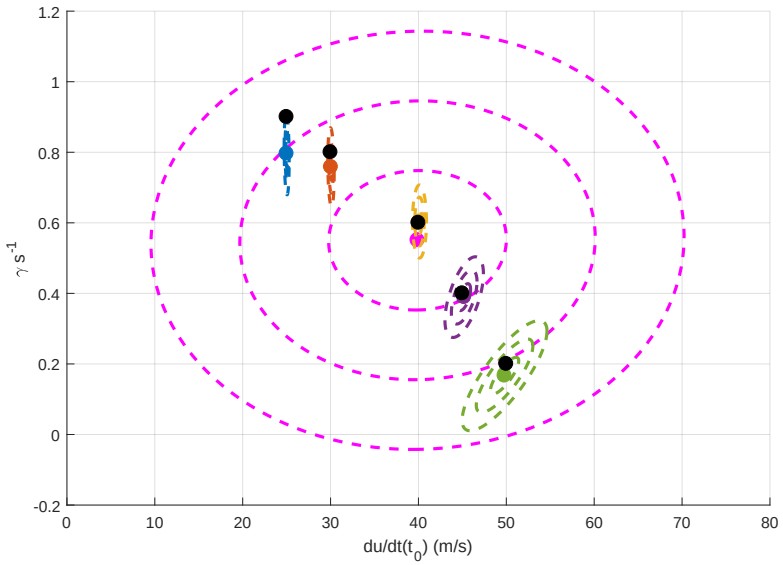


Figure 6.6: Prior and adjusted moments of the parameters $\xi^* = (\gamma, \dot{z}_0)^{\mathrm{T}}$; $\mathrm{E}\left[\xi^*\right]$ is shown as a magenta marker, with 1, 2 and 3 standard deviation contours shown in dashed magenta; the adjusted expectation $\mathrm{E}_z\left[\xi^*\right]$ for each case is plotted as a coloured marker, with colours corresponding to those in Figures 6.4 and 6.5, with associated uncertainty ellipses.

operation of the bells; there are numerous examples of churches where this motion causes the bells to behave unpredictably, making ringing difficult. Durham Cathedral is a good example of such a tower; it has a ring of 10 bells, with the heaviest weighing approximately 1500 kg, hung only metres from the top of its central tower. The motion of the tower and its effect on the ringing of the bells can be studied through numerical experiments. Smith and Hunt [2008] derive the equations of motion for the tower and the bells, solve the bell equations independently of the tower, and then use these solutions as a forcing for the tower equations, allowing them to study the effect on the motion of the tower of ringing the bells at different speeds and in different orders. This is a computationally efficient procedure which allows for simulation of the tower movement induced by the motion of the bells; however, it does not account for the bell-tower interactions. In order to model the joint motion of all components, we must numerically solve the system.

Other interesting aspects of the motion of a ringing bell are studied by Woodhouse et al. [2012]; they use a different system of equations to model the interaction between a swinging bell and its clapper, relating their model to experimental results and to the acoustic properties of the bell. Lund et al. [1995] carry out a study of the dynamic behaviour of a number of bell towers in the North-East of England, obtaining data by placing accelerometers in each one and ringing the heaviest bell.

In Section 6.2.1, we present the equations of motion for the coupled bell-tower model, and in Section 6.2.2, we outline the numerical scheme that we will use to approximate the solution. In Section 6.2.3, we outline our model for the numerical discrepancy, and describe the procedure for fitting it. Then, in Section 6.2.4, we use the procedure outlined in Section 6.1.6 to characterise the full prior distribution, and to update the components using observations. Finally, in Section 6.2.5, we discuss the results of this example, and the strengths and weaknesses of the framework in general, and we consider further work which might be done in this area. Some of the code used to implement the example from this section is supplied in Appendix F.

6.2.1 Equations of motion

As in Smith and Hunt [2008], we derive the equations of motion for the coupled bell-tower system from the its Lagrangian. We model the displacement of the tower in both the horizontal and vertical directions, denoting these values at time t by $x(t) = (x_1(t), x_2(t))^{\mathrm{T}}$; ignoring the effect of the bells, the Lagrangian for the tower is

$$\mathcal{L}_T = \sum_{i=1}^{2} \left[\frac{1}{2} M \dot{x}_i^2 - \frac{1}{2} \kappa_i x_i^2 \right] \tag{6.2.6}$$

where M is the mass of the tower, and κ_i is the tower spring constant in the i^{th} direction. Applying the Euler-Lagrange equation to determine the equation of motion and allowing for the damping of the motion, we obtain separate differential equations for each direction $i = 1, 2$

$$\ddot{x}_i + 2\lambda_i \dot{x}_i + \omega_i^2 x_i = 0$$

where λ_i is the damping constant in the i^{th} direction, and $\omega_i^2 = \kappa/M$ is the frequency of the motion.

For the bells, we denote the angle that bell i makes with the downward vertical direction at time t by $\theta_i(t)$, writing the collection as $\theta(t) = (\theta_1(t), \ldots, \theta_{n_\theta}(t))^{\mathrm{T}}$. Independently of the tower, the Lagrangian for the bells is

$$\mathcal{L}_B = \sum_{j=1}^{n_\theta} \left[\frac{1}{2} m_j r_{g_j}^2 \dot{\theta}_j^2 + \frac{1}{2} m_j r_{c_j}^2 \dot{\theta}_j^2 + m_j f_g \cos\theta_j \right] \tag{6.2.7}$$

where m_j is the mass of bell j, r_{c_j} is the distance from its pivot to its centre of mass, r_{g_j} is its radius of gyration and f_g is acceleration due to gravity. Applying the Euler-Lagrange equation gives

$$l_j \ddot{\theta}_j + f_g \sin\theta_j = 0$$

as the equation of motion for each bell, where $l_j = (r_{g_j}^2 + r_{c_j}^2)/r_{c_j}$.

Combining the Lagrangians (6.2.6) and (6.2.7), and modifying the kinetic energy of the bell, so that it receives a horizontal forcing from the motion of the tower, we

obtain the following Lagrangian for the coupled bell-tower system

$$\mathcal{L} = \sum_{i=1}^{2}\left[\frac{1}{2}M\dot{x}_i^2 - \frac{1}{2}\kappa_i x_i\right] + \sum_{i=1}^{2}\sum_{j=1}^{n_\theta} d_{ij} m_j r_{c_j}\dot{\theta}_j\dot{x}_i\cos\theta + \sum_{j=1}^{n_\theta}\left[\frac{1}{2}m_j l_j r_{c_j}\dot{\theta}_j^2 + m_j f_g r_{c_j}\cos\theta_j\right].$$

Applying the Euler-Lagrange equations to the joint Lagrangian, we obtain one equation of motion for each of the components of the tower motion, and one for each of the bells

$$\ddot{x}_i + 2\lambda_i\dot{x}_i + \omega_i^2 x_i = -\sum_{j=1}^{n_\theta} d_{ij}\frac{m_j r_{c_j}}{M}\left[\ddot{\theta}_j\cos\theta_j - \dot{\theta}_j^2\sin\theta_j\right] \tag{6.2.8}$$

$$l_j\ddot{\theta}_j + f_g\sin\theta_j = -\sum_{i=1}^{2} d_{ij}\ddot{x}_i\cos\theta_j \tag{6.2.9}$$

where $d_j = (d_{1j}, d_{2j})^{\mathrm{T}}$ is an orientation vector for bell j. These equations, supplemented with initial positions $\{x_i(t_0)\}$ and $\{\theta_j(t_0)\}$, and initial velocities $\{\dot{x}_i(t_0)\}$ and $\{\dot{\theta}_j(t_0)\}$ implicitly determine the motion of this system for all time.

Under this model, the motion of the tower is driven by the forces generated by the rotation of the bells, and the motion of the bells is driven by gravity and the forces generated by the motion of the tower. It is when the bell is at the top of its swing ($\theta_j = 180°$ or $\theta_j = -180°$) that the ringer experiences the effect of the tower movement; the ringing of methods requires the ringer to apply force to the bell through the rope at or close to the top of its swing, and so any unexpected force applied to the bell at this point must be compensated for by the ringer if the bell is to strike at the right moment.

The fixed parameter settings that we use for the bells are summarized in table 6.1. The bell weights and the swing directions are chosen to coincide with those of the bells at Durham Cathedral [Dove, 2015]; however, measurements of the other parameters for these bells are not available, so we simply use values taken from the 10 heaviest bells of the old ring at Great St Mary, Cambridge (provided in Smith and Hunt [2008]), which were of roughly the same weights as the bells at Durham. The mass of the tower is fixed at $M = (2 \times 10^4)$ kg. The other parameters of the tower motion (damping constants $\{\lambda_i\}$ and natural frequencies $\{\omega_i\}$) are assumed to be

	m					
Bell	cwt	qr	lb	r_c (mm)	l (mm)	Swing (handstroke)
1	6	1	27	407	628	N-S
2	7	0	8	405	665	S-N
3	7	3	10	417	665	S-N
4	7	3	14	429	712	S-N
5	9	3	25	417	752	S-N
6	11	0	8	428	790	W-E
7	12	2	8	403	856	W-E
8	15	3	1	438	887	W-E
9	21	2	9	451	984	W-E
10	28	0	6	559	1027	S-N

Table 6.1: Fixed bell parameters used as input to the model (6.2.9). The weights and layout are those of the bells at Durham Cathedral [Dove, 2015], and the other bell parameters are those of the old bells at Great St Mary, Cambridge (taken from [Smith and Hunt, 2008])

subject to uncertainty, and this is handled through our modelling in Sections 6.2.2 and 6.2.3.

6.2.2 Numerical scheme

We choose to use another Euler scheme (Section 6.1.1) to generate numerical approximations for this problem. Since this is a system of second-order equations, we must track the evolution of both the position and velocity components of the solution; we stack the components of the model into a $(4 + 2n_\theta)$-dimensional state vector as follows

$$u(t) = \begin{pmatrix} \dot{x}(t) \\ \dot{\theta}(t) \\ x(t) \\ \theta(t) \end{pmatrix}.$$

This allows us to stack the equations (6.2.8) and (6.2.9) for the acceleration and (trivial) equations for the velocity as follows

$$A\frac{du}{dt}(t) = b$$

where the matrix $A = A(u(t), \xi)$ and the vector $b = b(u(t), \xi)$ are partitioned according to the components of the state vector

$$A(u(t), \xi) = \begin{pmatrix} A_{\dot{x}\dot{x}} & A_{\dot{x}\dot{\theta}} & 0 & 0 \\ A_{\dot{\theta}\dot{x}} & A_{\dot{\theta}\dot{\theta}} & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{pmatrix} \qquad b(u(t), \xi) = \begin{pmatrix} b_{\dot{x}} \\ b_{\dot{\theta}} \\ b_x \\ b_\theta \end{pmatrix}$$

where $b_x = \dot{x}(t)$, $b_\theta = \dot{\theta}(t)$, and the elements of the remaining components can be read off from (6.2.8) and (6.2.9)

$$\begin{aligned} A_{\dot{x}_i\dot{x}_j} &= I_{ij} & A_{\dot{x}_i\dot{\theta}_j} &= d_{ij}\frac{m_j r_{c_j}}{M}\cos\theta_j \\ A_{\dot{\theta}_i\dot{x}_j} &= d_{ji}\cos\theta_i & A_{\dot{\theta}_i\dot{\theta}_j} &= l_i I_{ij} \\ b_{\dot{x}_i} &= \sum_{j=1}^{n_\theta} d_{ij}\frac{m_j r_{c_j}}{M}\dot{\theta}_j^2\sin\theta_j - 2\lambda_i\dot{x}_i - \omega_i^2 x_i & b_{\dot{\theta}_j} &= -f_g\sin\theta_j \ . \end{aligned} \tag{6.2.10}$$

Using this notation, we have that

$$\begin{aligned} \frac{du}{dt}(t) &= f(u(t), \xi) \\ &= A^{-1}b \end{aligned}$$

where $f(.)$ has no explicit time dependence, and so under our Euler scheme, the i^{th} component of the solution evolves as

$$\hat{u}_i(u(t_k), t_k, t_{k+1}, \xi) = u_i(t_k) + h_k f(u(t_k), \xi) \ .$$

6.2.3 Numerical discrepancy model

Having set up our numerical scheme, we choose a form for our numerical discrepancy model; since we have again chosen an Euler solver, we specify a basis for the discrepancy model based on the higher order terms in the power series expansion at

the initial point. We set the vector of basis functions as

$$g\left(\theta\right) = \begin{pmatrix} \frac{1}{2}h_k^2 \frac{d^2u}{dt^2}\left(\theta\right) \\ \frac{1}{6}h_k^3 \frac{d^3u}{dt^3}\left(\theta\right) \end{pmatrix}$$

where $\theta = \left\{u\left(t_k\right), t_k, t_{k+1}, \xi\right\}$ is the set of solver inputs at this point. The second-order derivatives are found directly as

$$\begin{aligned} \frac{d}{dt}\left(A\frac{du}{dt}\right) &= \frac{db}{dt} \\ \frac{dA}{dt}\frac{du}{dt} + A\frac{d^2u}{dt^2} &= \frac{db}{dt} \\ \frac{d^2u}{dt^2} &= A^{-1}\Big[\frac{db}{dt} - \frac{dA}{dt}\frac{du}{dt}\Big] \end{aligned}$$

and the third-order derivatives are also found directly by re-differentiating the second line of the above

$$\begin{aligned} \frac{d}{dt}\Big[\frac{dA}{dt}\frac{du}{dt} + A\frac{d^2u}{dt^2}\Big] &= \frac{d^2b}{dt^2} \\ \frac{d^2A}{dt^2}\frac{du}{dt} + 2\frac{dA}{dt}\frac{d^2u}{dt^2} + A\frac{d^3u}{dt^3} &= \frac{d^2b}{dt^2} \\ \frac{d^3u}{dt^3} &= A^{-1}\Big[\frac{d^2b}{dt^2} - \left(\frac{d^2A}{dt^2}\frac{du}{dt} + 2\frac{dA}{dt}\frac{d^2u}{dt^2}\right)\Big] \end{aligned}$$

where the components of the first and second derivatives of A and b are easily obtained from the expressions (6.2.10). For the residual process, we simply set

$$\mathrm{Cov}\left[r_i\left(\theta\right), r_j\left(\theta'\right)\right] = V_{ij} I(\theta = \theta') \ .$$

We choose not to impose a correlated residual structure in this instance because we judge that, for the time-step lengths and parameter settings that we will be using, the error introduced through neglecting the terms of higher than third-order from the power series will be too small for it to be worth us investigating its structure over its input space. Generating samples with errors small enough for structure of this magnitude to stand out using the procedure from 6.1.6 would be computationally expensive, as would using an emulator for our discrepancy model in place of a sample linear regression.

Using the above components, we run the analysis from Section 6.1.6 to build a prior model for the numerical discrepancy under our solver. We generate data by setting

$\mathrm{Var}\left[\eta_{x0}(h)\right] = (h \times 10^{-4})^2$ for the $\dot{x}$ and x components, and $\mathrm{Var}\left[\eta_{\theta 0}(h)\right] = (h \times 0.1)^2$ for the $\dot{\theta}$ and θ components; these values are selected so as to ensure that the numerical solution samples generated using the refined grid are always within 3 standard deviations of those generated using coarse grid, for all parameter settings of interest (see discussion in Section 6.1.6). We fix $\mathrm{E}\left[\beta_{ip}\right]$ and $\mathrm{Cov}\left[\beta_{ip}, \beta_{jq}\right]$ for all components by performing a linear regression using a sample of 200 points, and the marginal variance matrix V for the residual process is fixed to the empirical covariance of the residuals from this regression. The quality of this model is then assessed using an additional sample of 50 points, all of which are found to lie within 3 standard deviations of the predictive mean.

6.2.4 Results

Using the solver and numerical discrepancy specifications from Sections 6.2.2 and 6.2.3, we now characterise all of the components of the full model. For this example, we use the junction tree as a basis for analysis; to reduce the computational effort required, we also drop all edges between the numerical discrepancy components. On this basis, the junction tree contains three cliques for each time-step t_k:

- $\mathrm{Q}_1\left(t_k\right) = \{\xi^*, u\left(t_{k-1}\right), \hat{u}\left(t_k\right), \eta\left(t_k\right)\}$
- $\mathrm{Q}_2\left(t_k\right) = \{\xi^*, \hat{u}\left(t_k\right), \eta\left(t_k\right), u\left(t_k\right)\}$
- $\mathrm{Q}_3\left(t_k\right) = \{u\left(t_k\right), z_k\}$

and the tree is constructed by attaching $\{\mathrm{Q}_1\left(t_k\right), \mathrm{Q}_2\left(t_k\right)\}$ together in a chain for $k = 1, \ldots, n_t$, and then attaching $\mathrm{Q}_3\left(t_k\right)$ to $\mathrm{Q}_2\left(t_k\right)$ at each time-step. The cliques are then characterised in the order that they appear in the tree; a sample of 500 points is generated for each component (using a Gaussian distribution), and then these samples are used to compute the expectations of each component, and the full covariance structure within each clique. We use a solver grid of 1000 equal intervals, where $t_0 = 0$, $t_{1000} = 20$ and $h_k = 0.02$ for all steps. We initialise our prior sampler by specifying that $\mathrm{E}\left[\lambda_i\right] = 1$ and $\mathrm{Var}\left[\lambda_i\right] = (0.1)^2$, and that $\mathrm{E}\left[\omega_i\right] = 1.5$ and $\mathrm{Var}\left[\omega_i\right] = (0.2)^2$, both for $i = 1, 2$; all of the bell parameters and the tower mass

are fixed according to table 6.1.

Once the prior has been characterised in this way, we generate synthetic data for an adjustment by running the solver (including uncorrelated stochastic numerical discrepancy components) on a finer grid; we reduce the length of the time-step by a factor of 200, so that $h_k = 10^{-4}$ for the data generation. The components of the model are updated by using the structure of the junction tree to compute covariances between each data point and all other nodes (as outlined in Section 2.1.3, equation (2.1.9)).

Figures 6.7 and 6.8 show two different updates of the same prior specification; in Figure 6.7, we update using noise-corrupted observations of every 100^{th} time-point $(t_{100}, t_{200}, \ldots, t_{1000})$, and in Figure 6.8, we update using every 25^{th} time-point $(t_{25}, t_{50}, \ldots, t_{1000})$. In both cases, all components of the solution are observed at the selected knots.

In the prior trajectory (which is the same in both cases), we can clearly see the build-up of numerical uncertainty as the solution evolves in time. The marginal uncertainty about the tower position and velocity components is relatively much greater towards the end of the trajectory; the behaviour of these terms is much more irregular than that of the bell positions and accelerations, resulting in a much wider range of possible trajectories under the prior specification.

In the adjusted trajectories, we see the effect of the rich covariance structure imposed through the graphical prior specification. In Figure 6.7 we see that, despite the limited number of observations, we can make confident and accurate predictions for the solution values at a wide range of locations. This effect is particularly marked in the case of the North-South tower displacement component (Figure 6.7(d), top window); despite the limited number of measurements, we confidently and accurately predict the true tower displacement for all earlier times based on these measurements. Additionally, where our observations do not provide such strong information about the true trajectory, we see that our adjusted uncertainty specification reflects this; for example, from 18 to 20 seconds in Figure 6.8(c) (bottom window).

Considering Figure 6.8, we see that in this instance, the 40 observations are enough to make a fairly accurate assessment of the trajectories of all components across the

whole time period. Additionally, comparing, for example, Figures 6.7(c) and 6.8(c), we see that the numerical uncertainty that we have introduced to account for discretization error introduces quite wide variability in possible trajectory shapes for the tower components. We see from the way in which the adjusted moments adapt to the trajectory shape that the covariance structure imposed under our prior specification is flexible enough to be able to capture this variation in trajectory shape using only a small number of observations.

Model development Through this example, we have demonstrated the ability of the framework outlined in Section 6.1 to generate uncertainty specifications for complex ODE problems in which we can have confidence. In terms of the system under study though, the work in this chapter does not represent a serious attempt to quantify our uncertainty about the motion of the tower under a variety of different conditions, or the effect of this motion on the behaviour of the bells. Future improvements to the model could allow for a more detailed study to be undertaken. For example, we could adapt the model so that it includes a set of forcing terms which simulate the effect of the ringers applying forces to the bell through the rope, and so that these forces are applied in such a way as to simulate the ringing of methods.

In practice, the solution of the system in this way across the whole of a substantial piece of ringing would not be a sensible strategy; such a simulation would be very numerically intensive, and would provide a lot of superfluous detail. A better strategy would perhaps be to carry out an analysis using the procedure of Smith and Hunt [2008] and to use this to find particular ringing regimes which appear to generate large amounts of tower movement. The analysis described above could then be run using initial conditions based on these particular ringing regimes, to generate a more detailed picture of our uncertainties about the tower movement that we might see, and the effects that this might have on the difficulties experienced by the ringers due to this tower movement.

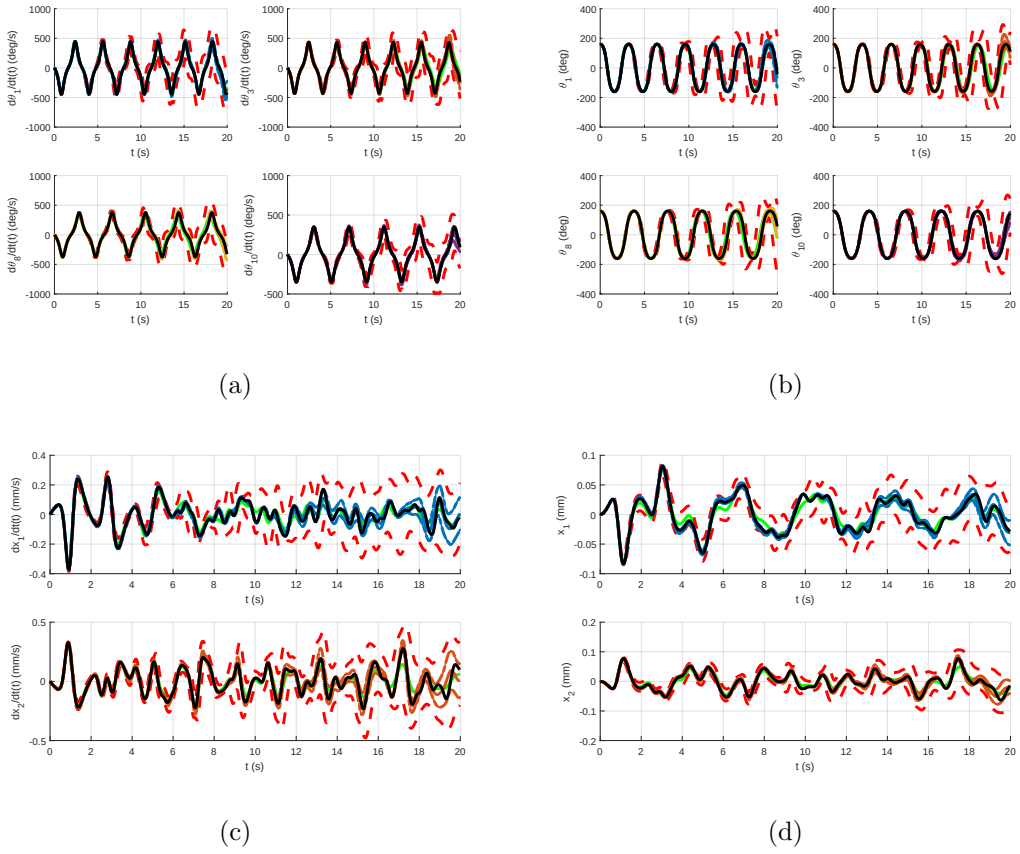


Figure 6.7: Plots of the prior and adjusted moments for the bell-tower model; first update case. Figure 6.7(a) shows the trajectory of $\dot{\theta}_i(t)$ for bells $i = 1, 3, 8, 10$, and Figure 6.7(b) shows $\theta_i(t)$ for the same four bells. Figure 6.7(c) shows both components $\dot{x}_i(t)$ of the tower velocity $(i = 1, 2)$, and Figure 6.7(d) shows the components of the tower displacement $x_i(t)$. In all cases, the prior expectation of the trajectory is shown in green, and three-standard deviation error bars are shown in dashed red; the adjusted moments are shown in coloured lines in each case, with a solid line for the adjusted mean and a dashed and dotted line for the three-standard deviation adjusted error bars. The trajectory from which the samples are actually generated is shown in black.

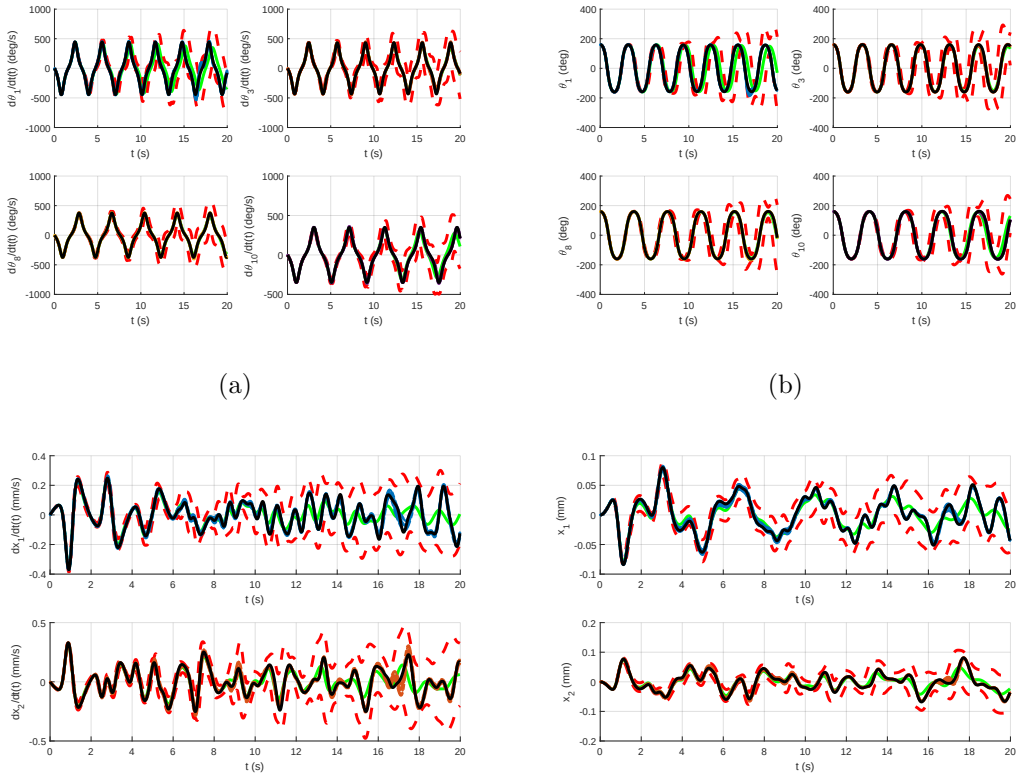


Figure 6.8: Plots of the prior and adjusted moments for the bell-tower model; second update case. The plots in each window are of the same quantities as the equivalent windows in Figure 6.7, with the same line styles and colours being used for the prior and adjusted moments, and for the actual trajectory.

6.2.5 Discussion

The modelling framework presented in this chapter, when coupled with a Bayes linear analysis, is an extremely useful tool in the study of ODE models for physical systems. Structuring the model according to the graph 6.1 allows us to produce a much richer covariance structure than it would have been possible to obtain by simply fitting an emulator to the output for the solver, and one which more closely represents the behaviour of the true solution. It also allows us to introduce a detailed model for the structure of the numerical discrepancy, at a level of accuracy appropriate for the solution effects which we are trying to assess. This rich covariance structure means that for many problems, we can accurately predict the solution over a substantial portion of its input space using only a small number of observations. We can also account for the effect of parameter uncertainty on solution uncertainty in a natural way, and use solution observations to learn about the parameters.

The use of a Bayes linear graphical model also provides access to computational tools which reduce the burden of assessing prior uncertainty about the trajectory induced by particular uncertainty specifications for the parameters and initial conditions, and of updating beliefs about the trajectory in the light of observations. We can store a limited prior specification on the graph, and then efficiently re-construct any covariance specification for a pair of non-adjacent nodes; additionally, constructing the junction tree corresponding to a particular graph allows us to adjust beliefs sequentially along the tree, eliminating the need to compute and store the large matrices required to characterise the full joint prior specification for all components.

The work presented in Sections 6.1 and 6.2 suggests a variety of different interesting areas for future research. First, the two examples presented represent the two most extreme cases in terms of belief separations for the numerical discrepancy; in the projectile trajectory example in Section 6.1.8, we impose a full prior correlation structure for the numerical discrepancy terms (in which each $\eta(t_k)$ has all the $\eta(t_l)$ as parents for $l < k$), whereas in the bell-tower example from Section 6.2, we do not impose any edges between these terms across time-steps. It would be interesting to explore (particularly in the bell-tower model) to what extent the imposition of a limited parent set (for example, including in the parent set of $\eta(t_k)$ only the

$\{\eta(t_{k-1}), \ldots, \eta(t_{k-q})\}$ for some q) allows us to improve the quality of the correlation structure while retaining the efficiency of computation on the junction tree.

More broadly, the graph quantification technique presented in Section 6.1.6 makes second-order assumptions for all components and then, when higher-order moment specifications are required for numerical integration steps (propagation of uncertainty through the solver and numerical discrepancy components), we assume a particular distribution and characterise this using the second-order terms. This of course makes the result sensitive to this choice of distribution. In the projectile example (Section 6.1.8), the numerical scheme and the regression component of the discrepancy involve only polynomial functions of the uncertain inputs, and in the bell-tower example, we could easily arrive at the same point by approximating any sines and cosines with low-order Taylor expansions. Within the Bayes linear framework, we need not limit ourselves to first- and second-order moments of any particular component; we can carry around first- and second-order moments for as many integer powers of the component as we like. In the future, then, it would be interesting to model solution trajectories with the same framework, but using a second-order specification for component powers up to a given order p. This has the potential to produce a framework in which we can study the behaviour of a trajectory without having to make any distributional assumptions, allowing us to investigate its behaviour under a much more general range of prior specifications for the components which actually drive its behaviour, without making any superfluous assumptions for those which don't, all while retaining the attractive computational properties of the Bayes linear framework and eliminating the need for sampling.

6.3 Partial differential equations

Partial differential equation (PDE) models are also extremely common in the physical sciences. A PDE relates different partial derivatives of a function to each other, and when supplemented with a set of initial and boundary conditions, implicitly defines a solution surface as a function of a number of input variables. The general PDE model which we will consider has the following form; the n_u-dimensional solution

surface $u(x,t) = \{u_1(x,t), \ldots, u_{n_u}(x,t)\}$ defined on the domain $(x,t) \in \Omega \times [t_0, T]$ is known to satisfy the following conditions

$$\begin{aligned} \frac{\partial u_i}{\partial t}(x,t) &= F_i\left(u, \{\frac{\partial u}{\partial x_i}\}, \ldots, t, \xi\right) + f_i(x,t) \text{ for } x \in \Omega, t \in (t_0, T] \\ u_i(x,t_0) &= v_i(x) \text{ for } x \in \Omega \\ u_i(x,t) &= w_i(x,t) \text{ for } x \in \partial\Omega, t \in (t_0, T] \end{aligned} \tag{6.3.11}$$

where $F(.)$ could be a function of u and all of its spatial derivatives (of any order), and $f(x,t)$ is a known source function.

PDE systems are generally even more difficult to solve than ODE systems, and so it is even rarer that we can solve one exactly; where we can find an algebraic solution, this generally requires us to assume particular, standard forms for the initial and boundary conditions, which may not be appropriate for our particular application. In most situations where we use a PDE model, then, we must use some sort of numerical scheme in order to approximate the solution; this too is a more complex task than in the ODE case, since we must approximate the evolution of the function between time-steps over the whole of its spatial domain.

Numerical solution of the system (6.3.11) allows us to make progress at the expense of introducing a discrepancy between the solution that we obtain, and the solution that we originally envisaged when we specified (6.3.11); we wish to build a model for the system under study which accounts for our uncertainty about the true solution due to this discrepancy, in addition to the other 'usual' sources of uncertainty. Modelling this numerical discrepancy in the PDE case is more difficult than in the ODE case, due to the fact that we must model its behaviour across the spatial input domain of the solution as well as over the temporal input domain. As is the ODE case though, the numerical discrepancy corresponding to a particular scheme is highly structured, exhibiting systematic behaviour across its input domain, something which we can again exploit when constructing our model. A potential additional complication in the PDE case is that implicit solvers are often used, owing to their greater stability; this can introduce additional complications around the propagation of numerical or parameter uncertainty through the solver.

Previous work in this area has been undertaken by Conrad et al. [2017], who perturb

the basis functions of a finite element numerical scheme in order to represent the uncertainty introduced through numerical approximation, and then demonstrate the need to account for this source of uncertainty by showing that failure to account for it can result in mis-estimation of the model parameters. Graepel [2003] represents uncertainty about the solution of linear PDE by applying the differential operator to a Gaussian process and then making observations of the forcings and boundary conditions. Related work has also been carried out by Lindgren et al. [2011], who derive a stochastic differential equation for which the solution is a Gaussian field, and then obtain a corresponding finite-dimensional Gaussian specification using a finite element approximation. Fuglstad et al. [2015] provide further computational details for this procedure, using a finite volume scheme. In the numerical analysis literature, much work has been done to derive error bounds for different types of scheme; see, for example, Eymard et al. [2000] and references therein.

In the remainder of this section, we propose a modelling framework, similar to the one outlined in Section 6.1, which builds a stochastic representation of the solution to the system (6.3.11) that accounts for all sources of uncertainty. In Section 6.3.1, we outline two common methods for deriving discrete approximations to PDE solutions; then, in Section 6.3.2, we focus on the finite element method, and consider the likely structure of the numerical discrepancy that will arise through its use. In Section 6.3.3, we propose a model structure similar to the one outlined in Section 6.1.4 which handles all of the types of uncertainty that we wish to account for. In Section 6.3.4, we illustrate this model through application to a simple example, in which we numerically approximate the solution of the diffusion equation in one spatial dimension. Finally, in Section 6.1.5, we discuss the PDE modelling framework which has been proposed, and we consider ways in which it might be improved in the future.

6.3.1 Numerical schemes

We consider two popular types of numerical schemes for PDE; finite element and finite volume schemes. Under both of these schemes, we divide the spatial input domain into a set of discrete elements (or volumes), we define a set of basis func-

tions over these elements, we represent the solution as a weighted combination of these basis functions, and we use a weak formulation of the PDE to derive a set of relationships between the basis weights and the initial and boundary conditions and forcings. For linear PDE (and some simple non-linear PDE), this procedure results in a system of relations which can then be inverted to find our approximation to the solution. For the remainder of this section, we consider only the solution PDE problems with a single output dimension, where $u(x,t) = u_1(x,t)$.

Finite element analysis In a general finite element analysis (see, for example, Iserles [2008], chapter 9), we seek a weak solution to the problem, which we denote by $\hat{u}$, which satisfies the following relation for all test functions s

$$\int_{t_0}^{T} \int_{\Omega} \Big[\frac{\partial \hat{u}}{\partial t} - F(\hat{u}, \ldots, t, \xi) - f(x,t) \Big] s(x,t)\, dV dt = 0 .$$

Instead of seeking a weak solution over the spatial and temporal input domains, we approximate the evolution of the solution in time using a finite difference step; under this approach we seek a sequence of functions $\hat{u}(x,t_0), \hat{u}(x,t_1), \ldots, \hat{u}(x,t_{n_t})$ which satisfy the relations

$$\int_{\Omega} \Big[\frac{1}{h_k} \big(\hat{u}(x,t_{k+1}) - \hat{u}(x,t_k) \big) - F(\hat{u}, \ldots, t_{k+1}, \xi) - f(x,t_{k+1}) \Big] s(x)\, dV = 0 . \tag{6.3.12}$$

To find a solution, we must specify particular forms for $\hat{u}$ and s; we assume that both are finite, weighted sums of known basis functions $\{\phi_i\}$

$$\hat{u}(x,t_k) = \sum_{i=1}^{n_\phi} \hat{u}_i(t_k)\, \phi_i(x) \qquad s(x,t_k) = \sum_{i=1}^{n_\phi} s_i(t_k)\, \phi_i(x) . \tag{6.3.13}$$

Under this assumption, the problem is now one of finding the set of coefficients $\{\hat{u}_i(t_{k+1})\}$ which satisfy (6.3.12) for all settings of the coefficients $\{s_i(t_{k+1})\}$, for known $\{\hat{u}_i(t_k)\}$ and ξ.

If we substitute the basis expansions (6.3.13) into (6.3.12), we obtain the following

$$\sum_i s_i(t_{k+1}) \Bigg[\sum_j \int_{\Omega} \Big[\frac{1}{h_k} \big(\hat{u}_j(t_{k+1}) - \hat{u}_j(t_k) \big) \phi_i(x)\, \phi_j(x) - \phi_i(x)\, F(\hat{u}, \ldots, t_{k+1}, \xi) - \phi_i(x)\, f(x,t_{k+1}) \Big] dV \Bigg] = 0 . \tag{6.3.14}$$

To satisfy this relation for all s parametrized in this way, the inner sum must vanish for all $s_i(t_{k+1})$. If we further assume that $F(.)$ is a linear function of u and its partial derivatives, then we can re-write (6.3.14) as a system of linear equations

$$K(\theta)\,\hat{u}(t_{k+1}) = D(\theta)\,\hat{u}(t_k) - b(\theta) + f(\theta) \tag{6.3.15}$$

where $\theta = \{t_k, t_{k+1}, \phi_1, \ldots, \phi_{n_\phi}, \xi\}$ is the set of solver inputs, and the elements of the components are

$$\begin{aligned} D_{ij}(\theta) &= \int_\Omega \frac{1}{h_k}\phi_i(x)\,\phi_j(x)\,dV \\ K_{ij}(\theta) &= D_{ij}(\theta) - \int_\Omega \phi_i(x)\,F\left(\phi_j, \{\frac{\partial \phi_j}{\partial x}\}, \ldots, t_{k+1}, \xi\right) dV \\ f_i(\theta) &= \int_\Omega f(x, t_{k+1})\,\phi_i(x)\,dV \end{aligned} \tag{6.3.16}$$

where the indices i, j vary over $1, \ldots, (n_\phi - 1)$ and $b(\theta)$ contains boundary information. To find the basis coefficients at time t_{k+1}, we simply solve the system (6.3.15)

$$\hat{u}(t_{k+1}) = A(\theta)\left[D(\theta)\,\hat{u}(t_k) - b(\theta) + f(\theta)\right]$$

where $A = K^{-1}$.

If the function $F(.)$ is non-linear in u or its partial derivatives, then we can still use the finite element procedure, but the system of equations that we must solve (the analogue of (6.3.15)) becomes non-linear. This means that, in order to generate numerical solutions, we must introduce an additional numerical approximation step in order to solve the system. For example, finite element schemes for the Euler and Navier-Stokes equations produce systems of quadratic equations.

Basis functions In order to carry out the above analysis, we must first choose the basis functions $\{\phi_i\}$; these should be selected so that the resulting numerical approximation is flexible enough to capture variability in the solution, and so that the integrals in (6.3.16) are easy to perform. In a finite element analysis, we impose a polygonal (e.g. cubic, tetrahedral) mesh on the domain Ω, and then choose a basis which consists of a set of compactly-supported polynomials defined on the individual mesh elements. Defining the basis in this way confers a number of advantages:

- For complex domains, choosing such a basis affords us a great deal of flexibility in designing our mesh so that we capture all of the important variability in the solution; for example, if we use a tetrahedral mesh, we can mesh at a high resolution in regions where we expect rapid changes in the solution value, or where we wish to model carefully, and then at a lower resolution in regions where we expect low variability, or where we are not as interested;

- In order to obtain the expressions (6.3.14), we must only evaluate simple, polynomial integrals;

- Since the polynomials defined on the mesh elements are supported only on a small region of the space, the resulting system of linear relations is sparse, which ensures that it can be solved relatively quickly, even in large problems.

Finite volume analysis A finite volume analysis works in a very similar way to a finite element analysis; we again seek a weak solution to the PDE problem, but in this instance, we do so by exploiting conservation laws. Following Eymard et al. [2000], we assume that the differential equation has the following 'conservation law' form

$$\frac{\partial u}{\partial t} = \nabla . \left(G \left(u, \{ \frac{\partial u}{\partial x_i} \}, \ldots, t, \xi \right) \right) + f(x,t) \ .$$

If we then integrate this equation over a volume V, and over a particular time-step, we trivially obtain

$$\int_{t_k}^{t_{k+1}} \int_V \left[\frac{\partial u}{\partial t} - \nabla . \left(G \left(u, \{ \frac{\partial u}{\partial x_i} \}, \ldots, t, \xi \right) \right) - f(x,t) \right] dV dt = 0 \ .$$

We can now integrate the first term in time, and we can apply the divergence theorem to the second term, obtaining

$$\int_V \left[u(x, t_{k+1}) - u(x, t_k) \right] dV = \int_{t_k}^{t_{k+1}} \int_{\partial V} \left[G \left(u, \{ \frac{\partial u}{\partial x_i} \}, \ldots, t, \xi \right) . n(x) \right] d\omega(x) \, dt + \int_{t_k}^{t_{k+1}} \int_V f(x,t) \, dV dt$$

where ∂V is the boundary of the volume V, $n(x)$ is the outward normal vector to the boundary at x, and $\omega(x)$ is the integration measure on the boundary. Under this formulation, the total change in u over the interval $[t_k, t_{k+1}]$ is equal to the total

flux of $G(.)$ over the boundary during this period, and the total forcing over this time interval. We use this relation to derive a discrete approximation to the PDE by meshing the domain into a set of volumes $\{V_i\}$ such that $\cup_i V_i = \Omega$ and $V_i \cap V_j = \emptyset$ for $i \neq j$ and introducing the average values

$$\hat{u}(t_k) = \frac{1}{m(V_i)} \int_{V_i} u(x, t_k)\, dV \qquad f(t_k) = \frac{1}{m(V_i)} \int_{V_i} f(x, t_k)\, dV$$

where $m(V_i)$ measures the volume of element V_i. The numerical scheme then implicitly defines the $\hat{u}(t_{k+1})$ in terms of the $\hat{u}(t_k)$ as

$$\hat{u}(t_{k+1}) + \Gamma(\hat{u}(t_{k+1}), t_k, t_{k+1}, \xi) = \hat{u}(t_k) + f(t_{k+1}) \tag{6.3.17}$$

where $\Gamma(.)$ approximates the values of $G.n(x)$ on the boundary elements of each volume V_i. As in the finite element case, we must solve the system of equations resulting from (6.3.17) to obtain $\hat{u}(t_{k+1})$ at each time-step. If the original PDE was linear, the resulting system will be linear; if the original PDE was non-linear, however, these equations will be non-linear, and so we must introduce some additional approximation if we are to solve them.

6.3.2 Numerical discrepancy: finite element schemes

By choosing to solve a PDE using a particular numerical scheme, we obtain an approximate solution which is discrepant from the true solution to the equation in some unknown way; in order to improve the quality of our predictions, we wish to assess the magnitude of of this discrepancy, and to model its structure across the spatial and temporal domains. As in the ODE case, we can further investigate the structure of the real discrepancy, and we can base our model around components which we believe make an important contribution. We discuss two ways of structuring the behaviour of the numerical discrepancy in the finite element case. First, we consider the Taylor expansion of the solution around the central point of the basis function, and build a model for the discrepancy in terms of the difference between the numerical approximation and the series representation at these points; then, we consider further imposing structure on the discrepancy within spatial elements by generating a further finite element approximation given the (uncertain) true values at the element boundaries

Discrepancy at the nodes Consider a PDE in one spatial dimension and time: if we select our basis functions so that, for any given point x, we have that $\sum_i \phi_i(x) = 1$ (this is generally the case for finite element basis functions), then we can trivially represent the real solution to (6.3.11) at time t_k as

$$u(x, t_k) = \sum_{i=1}^{n_\phi} u(x, t_k)\, \phi_i(x) \ .$$

If we now Taylor expand the solution around a point c_i in the centre of the support of each basis function, we can also write the solution as

$$u(x, t_k) = \sum_{i=1}^{n_\phi} \Big(u(c_i, t_k) + (x - c_i) \frac{\partial u}{\partial x}(c_i, t_k) + \frac{1}{2!}(x - c_i)^2 \frac{\partial^2 u}{\partial x^2}(c_i, t_k) + \dots \Big) \phi_i(x)$$

assuming that the series converges to the solution over the support of each basis function. We use this expression to motivate a model based on the high-order terms from this expansion; this is effectively a set of local polynomial regressions centred around the $\{c_i\}$, supported at the inputs x for which the corresponding basis functions are non-zero. Because the finite element scheme outlined in Section 6.3.1 represents the time evolution of the solution-basis function inner products, it is not guaranteed that the numerical solution $\hat{u}$ will agree with the true solution at the $\{c_i\}$ if we refine the temporal grid, and so our approximation at time t_k is

$$u(x, t_k) = \hat{u}(x, t_k) + \eta(x, t_k)$$

where $\hat{u}$ is the finite element approximation of the first n_{drv} spatial partial derivatives

$$\hat{u}(x, t_k) = \sum_{i=1}^{n_\phi} \Big(\sum_{p=0}^{n_{\text{drv}}} \frac{1}{p!}(x - c_i)^p \hat{u}_i^{(p)}(t_k) \Big) \phi_i(x)$$

and $\hat{u}_i^{(p)}(t_k)$ is our finite element approximation to $\frac{\partial^p u}{\partial x^p}(x, t_k)$ at c_i, and the discrepancy η is structured so as to ensure agreement with the real solution at the c_i

$$\eta(x, t_k) = \sum_{i=1}^{n_\phi} \Big(\sum_{p=0}^{n_{\text{drv}}} \frac{1}{p!}(x - c_i)^p \eta_i^{(p)}(t_k) \Big) \phi_i(x) + w(x, t_k)$$

where $w(.)$ is designed so that it vanishes at the c_i. If the low-order behaviour of the solution in the region around each c_i is approximated by the polynomial terms, then $w(.)$ should be designed to capture the behaviour of higher-order contributions

in the series.

For this framework to be useful, we must be willing to make an assessment of our uncertainty about the numerical discrepancy based on our knowledge of the solution behaviour. As in the ODE case, there are a number of different ways in which we could do this:

- At the simplest level, we could make an independent uncertainty specification for each of the $\eta_i^{(p)}(t_k)$ at each knot and time step;
- Being slightly more sophisticated, we could incorporate some of our beliefs about the structure of the discrepancy across space and time by imposing a simple covariance structure; for example, we could relate the value of the numerical discrepancy at time t_{k+1} to its value at time t_k, and we could assume a particular covariance structure across the spatial knots at a particular time-step;
- For a more detailed description, we could adopt a strategy analogous to the one detailed in Section 6.1.6 and fit a model to the discrepancy to be used as an input to our analysis of the solution; as before, we could generate approximate samples from the discrepancy by making a variance specification which scales with the coarseness of the mesh and running the solver at a higher resolution.

Within-element discrepancy Once we have specified a model for the numerical discrepancy at the centres of the basis functions, we must do the same for the continuous component $w(x, t_k)$; again, a number of options are open to us, depending on the level of effort that we wish to put into our model structure. Again, the simplest thing that we could do would be to assume that this error component is uncorrelated at each spatial location where we interrogate the solution. At the next level of complexity, we could impose a spatial stochastic process with some specified covariance function, taking care to ensure that this process vanishes at the centres of the basis functions.

If we wish to specify a more detailed model, we can further exploit the structure of

the solver: at the points $\{c_i\}$, for each time-step t_k, we have imposed that

$$\frac{\partial^p u}{\partial x^p}(c_i, t_k) = \hat{u}_i^{(p)}(t_k) + \eta_i^{(p)}(t_k) \ . \tag{6.3.18}$$

We can use this representation of the solution at the knots as the input to a further finite element solution. We sub-divide each of the intervals $[c_i, c_{i+1}]$ into a set of sub-elements, and solve on each of these intervals separately; for each interval, our initial conditions are taken from our representation of the solution at the previous time-step, and our boundary conditions are determined by the representation (6.3.18).

6.3.3 Bayesian analysis for numerical schemes

We combine the finite element scheme outlined in Section 6.3.1 with a model for the numerical discrepancy which uses some of the elements outlined in Section 6.3.2 to create a model for the true solution of the system (6.3.11) which accounts for all of our uncertainties; as in the ODE case, we use a graphical model to summarise our belief separation specifications. This graphical model will then again form the basis for a Bayes linear analysis, in which we characterise the prior by propagating uncertainty through the diagram, and then use the diagram as a mechanism for updating beliefs about the model components using observations on the system.

As in the ODE case, our specification is made in four stages. First, we make a limited prior specification of just the initial condition, boundary condition and parameter functions; then we choose a numerical scheme with which to generate our approximate solution. Based on this choice, we then specify a form for the numerical discrepancy model; finally we use these three components and the graph 6.9 to work through the components and characterise the full joint prior distribution of the model. Again, as in the ODE case, we can make either a fully probabilistic or a Bayes linear specification; in this case, however, we only consider quantification of the prior in the Bayes linear case. The issues that we must consider if we are to make a fully probabilistic specification instead are largely the same as those outlined in 6.1.5.

Limited prior specification For a PDE model, we must specify our prior beliefs about the initial condition function, the boundary condition function and any

(continuous or discrete) parameters of the equation. For both the initial condition $v(x)$ and the boundary condition $w(x,t)$, we must make specifications $\mathrm{E}[v(x)]$, $\mathrm{Cov}[v(x), v(x')]$ and $\mathrm{E}[w(x,t)]$, $\mathrm{Cov}[w(x,t), w(x',t')]$ for the whole of the relevant (continuous) input domains (Ω and $\partial\Omega \times (0,T]$ respectively).

The analysis can be simplified by choosing a simple form for both these functions; for example, if we choose to represent both using a weighted combination of basis functions, as we do for the numerical solution (equation (6.3.13)), then the integrals which we need to perform in order to link the initial state to the numerical solution $\hat{u}(x,t_1)$ at the first time step (e.g. equation (6.3.14)) will be no more complex than for any other time-step.

Numerical scheme and discrepancy We must also choose a particular numerical scheme and make a prior specification for our numerical discrepancy model. For the remainder of this section, we assume that we choose to use a finite element numerical scheme; much of the following would also apply if we were to choose a finite volume scheme instead. Our options for the numerical discrepancy model are discussed in Section 6.3.2; whatever choice we make for this model, we must be able to specify the moments $\mathrm{E}[\eta(x,t_k)]$ and $\mathrm{Cov}[\eta(x',t_k), \eta(x',t_k)]$ for each time-step, conditionally on the solution $u(x,t_{k-1})$ at the previous time-step or the numerical discrepancies at any previous times where required.

Characterising the full prior Once we have made the limited prior specification, selected a numerical solver and specified a form for the numerical discrepancy model, we can use the graph 6.9 to characterise the full joint prior distribution. At each time step t_k, we need to use our diagram to characterise three different components: the numerical approximation $\hat{u}(x,t_k)$, the numerical discrepancy $\eta(x,t_k)$ and the solution $u(x,t_k)$.

First, the numerical approximation at time-step t_k; recall that we use the basis representation (6.3.13). Given the components at all previous times, the expectations of the basis coefficients $\{\hat{u}_i(t_k)\}$ are as follows (using the Einstein summation

convention for indices)

$$\mathrm{E}\left[\hat{u}_i\left(t_k\right)\left|u\left(t_{k-1}\right),\xi^*\right.\right] = A_{ip}\left(\theta\right)\left[D_{pq}\left(\theta\right)u_q\left(t_{k-1}\right) - b_p\left(\theta\right) + f_p\left(\theta\right)\right] \tag{6.3.19}$$

where as before, θ is our abbreviation for the full set of solver inputs. The expectations of these terms independent of their parent nodes are therefore as follows

$$\mathrm{E}\left[\hat{u}_i\left(t_k\right)\right] = \mathrm{E}\left[\mathrm{E}\left[\hat{u}_i\left(t_k\right)\left|u\left(t_{k-1}\right),\xi^*\right.\right]\right] \tag{6.3.20}$$

where the outer expectation is taken with respect to $p\left(u\left(t_{k-1}\right),\xi^*\right)$, a distribution for which we must choose a form. The covariances between these components can be computed using the law of total covariance

$$\begin{aligned}\mathrm{Cov}\left[\hat{u}_i\left(t_k\right),\hat{u}_j\left(t_k\right)\right] = {}& \mathrm{E}\left[\mathrm{Cov}\left[\hat{u}_i\left(t_k\right),\hat{u}_j\left(t_k\right)\left|u\left(t_{k-1}\right),\xi^*\right.\right]\right] \\ & + \mathrm{Cov}\left[\mathrm{E}\left[\hat{u}_i\left(t_k\right)\left|\hat{u}\left(t_{k-1}\right),\xi^*\right.\right],\mathrm{E}\left[\hat{u}_j\left(t_k\right)\left|u\left(t_{k-1}\right),\xi^*\right.\right]\right] .\end{aligned}$$

Conditional on the parent components, the covariance between the coefficients is zero, so we only need to evaluate the second term as

$$\begin{aligned}\mathrm{Cov}\left[\hat{u}_i\left(t_k\right),\hat{u}_j\left(t_k\right)\right] = {}& \mathrm{E}\left[\mathrm{E}\left[\hat{u}_i\left(t_k\right)\left|u\left(t_{k-1}\right),\xi^*\right.\right],\mathrm{E}\left[\hat{u}_j\left(t_k\right)\left|u\left(t_{k-1}\right),\xi^*\right.\right]\right] \\ & - \mathrm{E}\left[\hat{u}_i\left(t_k\right)\right]\mathrm{E}\left[\hat{u}_j\left(t_k\right)\right] .\end{aligned} \tag{6.3.21}$$

The conditional expectation (6.3.19) is generally quite a complex object; it is a non-linear function (involving a matrix inverse) of the equation parameters ξ^*. For fixed ξ^*, we can easily evaluate the expectation (6.3.20) and the covariance (6.3.21) owing to the linearity of (6.3.19) in the $\{\hat{u}_i\left(t_k\right)\}$. These points are discussed further in Section 6.3.4.

For the numerical discrepancy, the predictions of $\eta\left(x,t\right)$ from our fitted model are dependent on the solution surface at the previous time-step and the parameter setting, and are generally correlated with the numerical discrepancy values at all previous time-steps. The exact structure of this correlation between time-steps will depend on the form that we choose for the numerical discrepancy (see the discussion in Section 6.1.2); we do not, therefore, explicitly outline the calculations that we need to perform in the general case. The procedure for uncertainty propagation, however, is the same as in the ODE case; we specify the expectation and covariance of $\eta\left(x,t_k\right)$

conditional on the parent set $\{\xi^*, \eta(x,t_1), \ldots, \eta(x,t_{k-1}), u(x,t_{k-1})\}$, and we compute the unconditional expectation $\mathrm{E}[\eta(x_i,t_k)]$ and covariances $\mathrm{Cov}[\eta(x_i,t_k), \eta(x_j,t_k)]$ for all locations of interest using the laws of total expectation and covariance.
Finally, we combine these components to determine the expectation and covariance of the solution at time t_k; its expectation is simply

$$\mathrm{E}[u(x_i,t_k)] = \mathrm{E}[\hat{u}(x_i,t_k)] + \mathrm{E}[\eta(x_i,t_k)]$$

and its covariance is

$$\begin{aligned}\mathrm{Cov}[u(x_i,t_k), u(x_j,t_k)] &= \mathrm{Cov}[\hat{u}(x_i,t_k), \hat{u}(x_j,t_k)] + \mathrm{Cov}[\eta(x_i,t_k), \eta(x_j,t_k)] \\ &\quad + \mathrm{Cov}[\hat{u}(x_i,t_k), \eta(x_j,t_k)] + \mathrm{Cov}[\eta(x_i,t_k), \hat{u}(x_j,t_k)] \ .\end{aligned}$$

Relationship to the system The relationship between the PDE solution surface $u(x,t_k)$ at the system $y(x,t_k)$ that it is designed to represent is the same as in the ODE case; we specify our beliefs about the discrepancy $\delta(x,t)$ and the measurement error ϵ, and we add on these components (which, as usual, are assumed to be uncorrelated with each other, and with the solution) to determine our beliefs about the system value and the observations respectively.

6.3.4 Example: diffusion equation

We now apply the model outlined in Section 6.3.3 to a simple example. The diffusion equation in one spatial dimension is

$$\begin{aligned}\frac{\partial u}{\partial t}(x,t) &= \kappa \frac{\partial^2 u}{\partial x^2}(x,t) + f(x,t), \text{ for } x \in \Omega, t \in (0,T] \\ u(x,t_0) &= v(x), \text{ for } x \in \Omega, t = t_0 \\ u(x,t) &= w(x,t), \text{ for } x \in \partial\Omega, t \in (0,T] \ .\end{aligned} \tag{6.3.22}$$

This is a standard model for several different physical systems; for example it is used to describe the conduction of heat in a particular medium, and the diffusion of particulate matter. It is a particularly useful example to study here because it has has an analytic solution for particular input domains and initial and boundary condition functions. If we assume that the initial condition function $v(x)$ and the

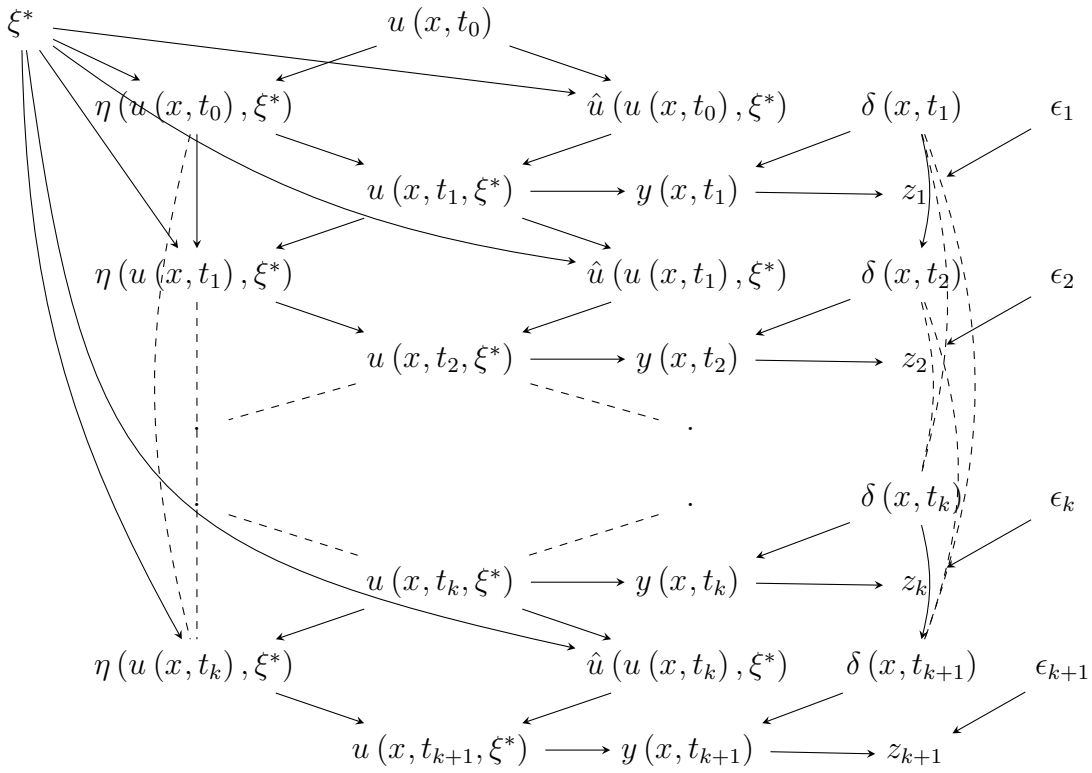


Figure 6.9: DAG representing the structure of the PDE model

forcing $f(x,t)$ have Gaussian forms, and we solve on $\Omega = (-\infty, \infty)$, $t \in [0, \infty]$, then we can evaluate the real solution by performing only a simple numerical integration in t; the solution under this specification is computed in Appendix C.1. Of course, using the finite element method, we can only solve over finite spatial and temporal domains. Therefore, for the remainder of our analysis, we fix $\Omega = [-2, 2]$, and fix $w(x,t)$ to be the real solution (obtained as in the appendix) on the boundary of this domain; we solve for $t \in [0, 2.5]$.

Numerical scheme For this problem, we use a finite element solver in which we track the behaviour of the solution and its first and second derivatives. Using the linearity of the PDE, it is easy to see that any derivative of the solution also evolves according to a diffusion equation, forced using the corresponding derivative of $f(x,t)$, i.e. for the p^{th} solution derivative, we have

$$\frac{\partial}{\partial t}\left(\frac{\partial^p u}{\partial x^p}(x,t)\right) = \frac{\partial^2}{\partial x^2}\left(\frac{\partial^p u}{\partial x^p}(x,t)\right) + \frac{\partial^p}{\partial x^p} f(x,t)$$

where the initial and boundary conditions for the derivative are found easily by differentiation. We focus on the solution and its spatial partial derivatives up to second order; as discussed in Sections 6.3.2 and 6.3.3, we assume the following form for our numerical approximation to the p^{th} spatial partial derivative solution $(p = 0, 1, 2)$ at time t_k

$$\hat{u}^{(p)}(x, t_k) = \sum_{i=1}^{n_\phi} \Big[\sum_{q=p}^{2} \frac{\hat{u}_i^{(q)}(t_k)}{(q-p)!} (x - c_i)^{q-p} \Big] \phi_i(x) \ .$$

Substituting the form of the diffusion equation into the expression (6.3.14), we find that our finite element approximation should satisfy the following system of equations

$$\sum_i s_i(t_{k+1}) \Bigg[\sum_j \int_\Omega \Big[\frac{1}{h_k} \phi_i(x) \big(\hat{u}^{(p)}(x, t_{k+1}) - u(x, t_k) \big) \\ - \phi_i(x) \frac{\partial^2}{\partial x^2} \big(\hat{u}^{(p)}(x, t_{k+1}) \big) - \phi_i(x) f(x, t_{k+1}) \Big] dV \Bigg] = 0$$

for $i, j = 1, \ldots, n_\phi$ and $p = 0, 1, 2$. Inputting the expression for each of the derivatives $\hat{u}^{(p)}(x, t_k)$, we obtain a system of equations of the same form as (6.3.15)

$$K(\theta) \hat{u}(t_{k+1}) + b(\theta) - f(\theta) = \bar{u}(t_k)$$

where in this instance, we use $\hat{u}(t_k) = (\hat{u}^{(0)}(t_k)^{\mathrm{T}}, \hat{u}^{(1)}(t_k)^{\mathrm{T}}, \hat{u}^{(2)}(t_k)^{\mathrm{T}})^{\mathrm{T}}$ to denote the vector which consists of the vertically-stacked numerical solution values at the nodes for all derivative orders. In this form, the matrix K has the following block structure

$$K = \begin{pmatrix} K^{(0)} & K^{(1)} & K^{(2)} \\ 0 & K^{(0)} & K^{(1)} \\ 0 & 0 & K^{(0)} \end{pmatrix}$$

where the elements of the matrix $K^{(p)}$ are

$$K_{ij}^{(p)} = D_{ij}^{(p)} - \int_\Omega \phi_i(x) \frac{\partial^2}{\partial x^2} \big((x - c_j)^p \phi_j(x) \big) dx \\ = D_{ij}^{(p)} + \int_\Omega \frac{\partial}{\partial x} \big(\phi_i(x) \big) . \frac{\partial}{\partial x} \big((x - c_j)^p \phi_j(x) \big) dx$$

where the we have applied the divergence theorem and assumed that the basis functions vanish on the boundary, and the elements of $D_{ij}^{(p)}$ are

$$D_{ij}^{(p)} = \int_\Omega \phi_i(x) (x - c_j)^p \phi_j(x) \, dx$$

for $i, j = 1, \ldots, (n_\phi - 1)$. The vectors $f(t_k)$ and $\bar{u}(t_k)$ are

$$f(t_k) = \begin{pmatrix} f^{(0)}(t_k) \\ f^{(1)}(t_k) \\ f^{(2)}(t_k) \end{pmatrix} \qquad \bar{u}(t_k) = \begin{pmatrix} \bar{u}^{(0)}(t_k) \\ \bar{u}^{(1)}(t_k) \\ \bar{u}^{(2)}(t_k) \end{pmatrix}$$

where

$$\begin{aligned} \bar{u}_i^{(p)}(t_k) &= \int_\Omega \frac{\partial^p u}{\partial x^p}(x, t_k)\, \phi_i(x)\, dV \\ f_i^{(p)}(t_k) &= \int_\Omega \left(\frac{\partial^p}{\partial x^p} f(x, t_{k+1}) \right) \phi_i(x)\, dV \ . \end{aligned}$$

The forcing components are computed by integrating the forcing function against the basis functions numerically (see appendix C.1 for details of the forcing function used); the solution at time t_k includes a contribution from the numerical discrepancy term, and so the computation of the $\bar{u}_i(t_k)$ is discussed further in the next paragraph.

For the basis function, we assume the common 'hat function' form [Iserles, 2008]; we divide the domain into n_ϕ elements by specifying a set $x_0 < x_1 < \ldots < x_{n_\phi}$ of monotonically increasing knots, and we specify that

$$\phi_i(x) = \begin{cases} \frac{(x - x_{i-1})}{d_i} & \text{for } x_{i-1} \leq x < x_i \\ \frac{(x_{i+1} - x)}{d_{i+1}} & \text{for } x_i \leq x < x_{i+1} \\ 0 & \text{else.} \end{cases} \tag{6.3.23}$$

Note that for this choice of basis function, the central point of the function ϕ_i is the knot x_i, and so $c_i = x_i$. The integrals for the solver matrices D and K are computed in appendix C.2.

Numerical discrepancy For the numerical discrepancy, we adopt the form discussed in Section 6.3.2. The numerical discrepancy for the p^{th} order derivative of the solution at time t_k is assumed to have the following form

$$\eta^{(p)}(x, t_k) = \sum_{i=1}^{n_\phi} \left(\sum_{q=p}^{2} \frac{1}{(q-p)!} (x - c_i)^{q-p} \eta_i^{(q)}(t_k) \right) \phi_i(x) + w^{(p)}(x, t_k) \ .$$

As discussed in Section 6.3.2, this form for the discrepancy is designed so that the knot x_i, we have that $u(x_i, t_k) = \hat{u}_i^{(p)}(t_k) + \eta_i^{(p)}(t_k)$; that is, the sum of the finite

element solution and the discrete discrepancy component agrees with the solution at each x_i. For this example, we choose a further basis expansion for the $w^{(p)}(.)$

$$w^{(p)}(x, t_k) = \sum_{i=1}^{n_\phi} \sum_{j=1}^{n_\gamma} w_{ij}^{(p)}(t_k) \gamma_{ij}(x)$$

where the basis functions $\gamma_{ij}(x)$ are defined for each element $[x_{i-1}, x_i]$ by dividing into 10 sub-elements and specifying a new set of 'hat' basis functions like (6.3.23). We must now make a (state-dependent) uncertainty specification for each of the components of this numerical discrepancy model. For the discrepancies $\{\eta_i^{(p)}(t_k)\}$ for each of the derivative states at the nodes, we do this by fitting a regression model. We generate approximate discrepancy data by evaluating the true solution at two points in time t_s and t_f, for which $0.05 \leq (t_f - t_s) \leq 0.2$, evolving over this time-step using the finite element scheme, and then subtracting the numerical solution from the true solution at the nodes. We then use this discrepancy data to fit joint regression model for the derivative states, which has the following form

$$\eta_i^{(p)}(t_f) = \sum_q \beta_{pq} g_q\left(u(x, t_s), x_i\right) .$$

The basis functions g are based on a local finite-difference solution over the interval $[x_{i-1}, x_{i+1}]$; the interval is divided into 20 elements, and the solution at t_f is re-approximated by using a 10-step finite difference scheme to evolve from t_s. The solution on the boundary at each time-step is simply approximated using the value here at the initial time. The finite-difference approximations on this refined grid (for all derivative states $p = 0, 1, 2$) form the basis functions for the regression model.

Quantifying the diagram Having specified the form of all of the model components, we compute our full joint prior specification by sampling the graph 6.9. First, we make a second-order prior specification for the initial state; we specify that $\mathrm{E}\left[u^{(p)}(x_i, t_0)\right] = u^{(p)}(x_i, t_0)$, fixing the expectation of the initial state to its known true value at the knots, and we introduce uncertainty by specifying that $\mathrm{Var}\left[u^{(p)}(x_i, t_0)\right] = (0.1\mathrm{E}\left[u^{(p)}(x_i, t_0)\right])^2$, so that the standard deviation of the initial state is proportional to its expected value. The parameter set $\xi = \kappa$ consists of only the diffusion constant; in this example, we simply fix $\kappa = 0.01$ and we do not

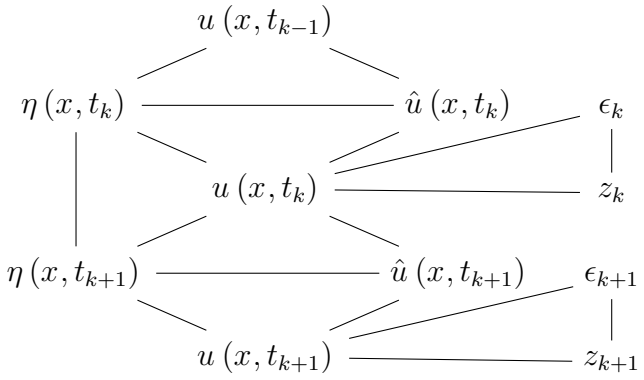


Figure 6.10: Undirected graph for the model components at times t_k and t_{k+1}.

handle best input uncertainty through the model, focussing only on initial condition and numerical discrepancy uncertainty.

Having specified a prior for the initial state, we quantify the remainder of the diagram by sampling; throughout this procedure, all samples are generated by using the relevant moments to characterise a multivariate Gaussian distribution. We characterise the solution surface at 25 evenly-spaced points in time, with $t_1 = 0.1, t_2 = 0.2, \ldots, t_{25} = 2.5$. We begin by sampling an initial solution state from the prior specification outlined above; we then work our way through the graph, at each stage generating a finite element solution $\hat{u}\,(x, t_{k+1})$ and sampling a numerical discrepancy $\eta\,()$ starting from the state $u\,(x, t_k)$, and then re-combining these to form $u\,(x, t_{k+1})$. For all three of these components, we store samples of the component values at the finite element nodes (for derivative states $p = 0, 1, 2$), and of component values at 50 evenly spaced-points across the spatial input domain.

The samples generated from this procedure are then used to characterise the cliques of the undirected graph shown in Figure 6.10. Note that for this example, we specify that only the numerical discrepancy components corresponding to adjacent time-steps are joined by an edge. This graph has the following cliques for each time-step:

- $\mathrm{Q}_1\,(t_k) = \{u\,(x, t_{k-1})\,, \hat{u}\,(x, t_k)\,, \eta\,(x, t_k)\,\}$;
- $\mathrm{Q}_2\,(t_k) = \{\hat{u}\,(x, t_k)\,, \eta\,(x, t_k)\,, u\,(x, t_k)\,\}$;
- $\mathrm{Q}_3\,(t_k) = \{\eta\,(x, t_k)\,, u\,(x, t_k)\,, \eta\,(x, t_{k+1})\,\}$ (for $k < 25$);

- $\mathrm{Q}_4\left(t_k\right) = \{u\left(x, t_k\right), \epsilon_k, z_k\}$.

The prior moments for the solution surface at a subset of the time knots are shown in Figure 6.11.

Adjustment Having characterised the full prior, we use the junction tree to adjust our prior beliefs using noise-corrupted measurements of the true solution at two of the time-steps. The solution is observed at all 50 spatial locations for which the solution was characterised in the above procedure at time-steps t_5 and t_{20}; all observations were generated by numerically evaluating the true solution (using the procedure in appendix C.1) and adding on Gaussian random noise with $\mathrm{Var}\left[\epsilon_{ik}\right] = (0.001)^2$. The adjustment is carried out for each set of observations in turn, by passing a covariance function around the cliques of the junction tree as described in Section 2.1.3.

The adjusted moments of the solution (for the same time-points as shown in Figure 6.11) are shown in Figure 6.12. As in the ODE examples of Sections 6.1.8 and 6.2, we see that the rich prior covariance structure that we imposed (through knowledge of the structure of the problem) allows us to make confident and accurate predictions for the solution at all time-steps based on a relatively small number of observations.

6.4 Discussion

In this chapter, we considered the sources of uncertainty that we encounter when representing the real world using a system of differential equations, and we proposed a Bayesian framework within which all of these sources of uncertainty can be handled. In particular, we introduced a model to describe the structure of the numerical discrepancy across time-steps, and proposed a procedure for approximately generating samples from the numerical discrepancy which can then be used to determine an appropriate prior specification for these components within the full framework. The graphical representations 6.1 and 6.9 provide a compact and easily interpretable representation of the flow of information between the components of the model, and can also form the basis for efficient sampling algorithms in the fully probabilistic

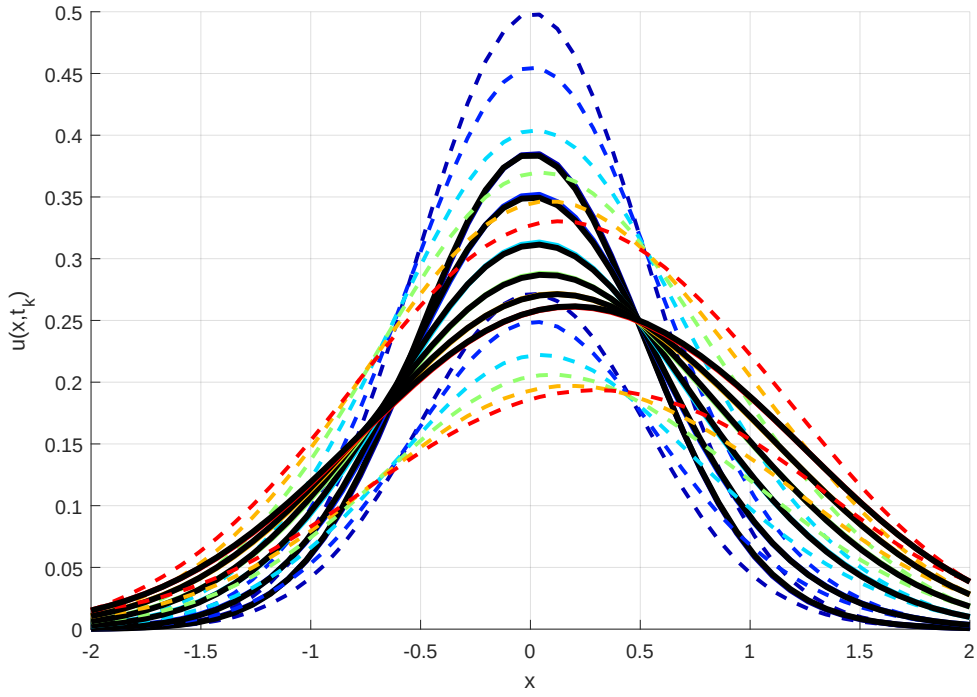


Figure 6.11: Prior moments of the solution surface $u(x, t_k)$ for time knots $\{t_2, t_5, t_{10}, t_{15}, t_{20}, t_{25}\}$ at 50 evenly-spaced spatial knots: the prior mean $\mathrm{E}[u(x, t_k)]$ is shown as a solid coloured line, and three-standard deviation error bars $\mathrm{E}[u(x, t_k)] \pm 3\mathrm{Var}[u(x, t_k)]^{1/2}$ are shown as dashed coloured lines. The true solution in each case is shown as a solid black line.

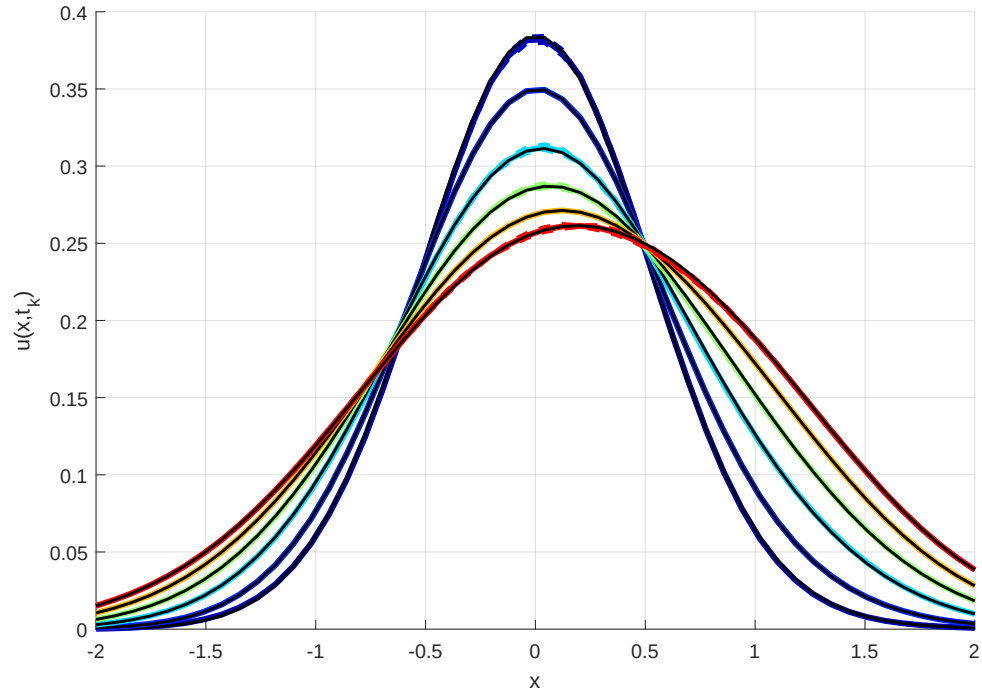


Figure 6.12: Adjusted moments of the solution surface at the same time points as in Figure 6.11: the line styles and colours used correspond across these two figures.

case, and efficient adjustment in the Bayes linear case.

We considered the Bayes linear case in more detail; if we make a second-order specification for the initial state, parameters and numerical discrepancy, and then generate the corresponding full prior specification using the graph (making distributional assumptions where required), we must only compute and store expectations and variances for each node and covariances corresponding to the edges. Then, if we find the junction tree corresponding to the original graph, we can adjust by each data point that we receive by performing a single pass through the graph, propagating the adjustment through the cliques as we reach them. Adjusting in this way is efficient in terms of both computation time and memory usage; for large problems, we can reduce the computational effort required yet further by reducing the number of links between numerical discrepancy components across time-steps.

The work in this chapter suggests a wide range of potential avenues for future research; some of these have already been discussed in Section 6.2.5, specifically in relation to the work on ODE problems. For this problem, future research should

focus on the propagation of uncertainty through the solver function and numerical discrepancy model; by specifying and propagating first- and second-order moments for higher powers of the solution, we can handle ODE solvers which have polynomial dependence on the solution at the previous time-step exactly, and we can handle other types of non-linear evolution function by introducing truncated series expansions for the complex terms.

For the PDE problem, many of the same areas could be investigated. For the example in Section 6.4, we considered only a simple, one-dimensional linear PDE; in practice, however, many of the PDE problems that we encounter will contain non-linear terms, and will have non-linear dependence on parameters. These non-linearities in the equation will induce non-linearities in the system of discrete equations that we obtain from the implicit finite element scheme outlined in Section 6.3.1 (or in any of the other explicit or implicit schemes that we might use), meaning that we cannot solve the system directly. In the non-stochastic case, an iterative numerical method is usually employed to find a solution to the system; within the framework described in Section 6.3 though, we could simply expand our prior specification so that it contains first- and second-order moments for all of the powers of the solution that we will encounter. This would then allow us to solve the system of equations resulting from (6.3.14) as a linear system in these powers, and to propagate our uncertainty directly.

Additionally, we could introduce an additional layer of modelling to improve our ability to handle problems with a large number of discrete elements. The use of a large number of elements results in a large system of equations (6.3.14), which means that we must invert a large matrix K for each time step. For compactly-supported basis functions, this matrix will be sparse, but for irregular mesh structures, inversion can still be a computational challenge. Additionally, the fact that we must use a numerical procedure to invert the matrix means that the only way to propagate uncertainty through the calculation is through sampling. Instead of sampling the matrix A for each ξ^* setting, we could emulate A or the product $Au\left(t_k\right)$ and then propagate our uncertainty through the emulator. This procedure would potentially save us computation time and increase accuracy; because of the sparse structure of

the matrix K induced by the neighbourhood structure of the mesh, we can fit an accurate emulator to an element of A using only a limited number of neighbours.

More generally, further work in this area could focus on developing statistical frameworks for handling other types of numerical schemes for PDE. A framework similar to the one outlined in Section 6.3.3 could be developed for the finite volume scheme discussed in Section 6.3.1, though in this case, it is perhaps less easy to imagine how we might handle uncertainty over the spatial input domain, owing to the use of a constant solution approximation within each element. It may also be worth investigating the properties of explicit schemes (finite element, finite volume or finite difference); solving in this way would perhaps reduce computation time per step and make it easier to track uncertainty through the components, at the expense of reducing the stability of the scheme, and the length of the time-steps that we are able to take.

An altogether different type of solver which may lend itself to treatment within a statistical framework is the smooth particle hydrodynamics method (see, for example, Rosswog [2009] or Monaghan [2005]). Within this framework, the initial condition is discretized by specifying its value at a set of 'particles' at known positions within the spatial input domain; the continuous solution is then approximated by smoothing using an interpolant. Popular choices of interpolating function include the cubic spline and the Gaussian kernel. These particle positions are then evolved in time according to the PDE; discretizing in this way results in a system of ODEs for the particle trajectories, for which a statistical analysis of the kind discussed in Section 6.1 could be performed. This kind of scheme is perhaps a more natural choice for PDE describing the behaviour of fluids (e.g. Euler, Navier-Stokes equations).

Appendix A

Notation

This appendix collects all of the notation defined throughout the thesis, and provides a reference to the place in the thesis where it is first defined. The notation is grouped into sections corresponding to those in the main body of the thesis; for each item of notation, we provide a reference to its first occurrence in the main text.

A.1 Atmospheric modelling

Symbol	Description	Introduced
$u\,(x, \dots)$	Scalar (mass) concentration of a gas species as a function of inputs (generally spatial location x, and others)	3
x	Three-dimensional spatial location (components $x = (x_x, x_y, x_z)^{\mathrm{T}}$)	3
$w\,(x, \dots)$	Three-dimensional wind vector (as a function of spatial location, and possibly other parameters; components $w = (w_x, w_y, w_z)^{\mathrm{T}}$)	4
$\omega(x, g, w)$	Three dimensional normalised downwind distance (components $\omega = (\omega_x, \omega_y, \omega_z)^{\mathrm{T}}$)	7
$a\,(\omega, w, g)$	Gaussian plume coupling coefficient, calculated as a function of downwind distance ω, wind w and source parameters g	7

ψ	Scalar source emission rate (m^3/s) for a point source	7
D	Atmospheric boundary later height (metres)	7
κ_y, κ_z	Squared standard deviation parameters of the Gaussian plume, in the horizontal and vertical directions	7

A.2 Bayesian analysis

A.2.1 Bayes linear analysis

Prior and adjusted expectations

Symbol	Description	Introduced
$\mathrm{E}[B]$	Expectation of the collection of quantities B.	25
$\mathrm{Var}[B]$	Variance matrix for the collection B. For a p dimensional quantity, $\mathrm{Var}[B]$ is a $p \times p$ matrix.	25
$\mathrm{Cov}[B, C]$	Covariance between two distinct sets of quantities B and D. Note that when $B = C$, $\mathrm{Cov}[B, B] = \mathrm{Var}[B]$.	25
$\mathrm{E}_D[B]$	Adjusted expectation for the collection B, given observation of the collection D.	26
$\mathrm{Var}_D[B]$	Adjusted variance for the collection B, given observation of the collection D.	27
$\mathrm{Cov}_D[B, C]$	Adjusted covariance between the collections B and C, given observation of the collection D.	27

Bayes linear graphical models

Symbol	Description	Introduced
$\mathrm{Pa}(B_i)$	Parent set of node B_i in a DAG.	27
$\mathrm{Ch}(B_i)$	Child set of node B_i in a DAG	27
Q_i	Clique i on an undirected graph.	30

A.2.2 Emulation of complex functions

Symbol	Description	Introduced
θ	General input set for a complex function. For later calculations, we will split $\theta = \{a, b\}$; see Section A.2.3	32
$f_i(\theta)$	Vector-valued output of a complex function run at input setting θ.	32
$g_i(\theta)$	Vector of basis functions for the regression component of the emulator.	32
β_{ij}	Basis coefficients for the regression component of the emulator.	32
$r_i(\theta)$	Correlated residual process at input setting θ.	32
$k_{ij}(\theta, \theta')$	Covariance function; determines the covariance between $r_i(.)$ and $r_j(.)$ evaluated at inputs θ and θ' respectively.	32
F_{ij}	Function data obtained at known input settings $F_{ij} = f_i(\theta_j)$; used to adjust prior moments.	34

A.2.3 Applications

Uncertainty analysis

Symbol	Description	Introduced
$\{a, b\}$	We split the full input set into $\theta = \{a, b\}$; in this context a are simulator-specific inputs, and b are system inputs.	43
z_{ij}	Data observed on a system under study; z_{ij} is the j^{th} observation on output i	43
$y_i(b)$	Output i of the system under study, evaluated at system inputs b	43
$\epsilon_i(b)$	Measurement error process for output i, evaluated at system input b	43

a^*	Best setting of the simulator-specific inputs a; evaluating the simulator $f(.)$ at this setting is sufficient for our inferences about the system value.	44
$\hat{f}_i(b)$	$\hat{f}_i(b) = f_i(a^*, b)$ Simulator evaluated at its best input setting	44

Inference for derivatives and integrals

Symbol	Description	Introduced
$\{a, b\}$	We split the full input set into $\theta = \{a, b\}$; here, we compute expectations of the function over $p(a)$, as a function of b.	51
$\bar{f}_i(b)$	Expectation of the function $f_i(a, b)$ with respect to $p(a)$	51

A.3 Experimental design

A.3.1 Design calculations

Symbol	Description	Introduced
q	Model parameters which we wish to infer from observations on the system under study	78
a	Decisions which we must make about the system	78
$L(a, q)$	Loss function describing the consequences of our decisions a upon realisation of model parameters q	78
$c_j(d_j)$	Function describing the cost of performing an experiment at design input setting d_j at stage j	92
z	Observations made on the system	82
d	Design inputs which we must select in order to run an experiment on the system	82

w	External inputs which affect the prediction for the system, but which cannot be controlled during experimentation	82
$\rho\left[z, w, d\right]$	Risk from an optimal decision	78

A.3.2 Sequential design calculations

Symbol	Description	Introduced
$L_j\left(a_j, q\right)$	Loss function which determines the consequences of decisions a_j upon realisation of model parameters q	92
$z_{[j]}$	The collection $\{z_1, \ldots, z_j\}$ of observations made on the system at all j stages.	92
$w_{[j]}$	The collection $\{w_1, \ldots, w_j\}$ of all external inputs governing the behaviour of the model for all j stages	92
$d_{[j]}$	The collection $\{d_1, \ldots, d_j\}$ of all design inputs for the experiments on the system carried out at all j stages	92
$\rho_j^t\left[z_{[j]}, w_{[j]}, d_{[j]}\right]$	The risk from an optimal terminal decision at stage j.	94
$\rho_j\left[z_{[j]}, w_{[j]}, d_{[j]}\right]$	The overall risk at stage j, taking into account the risk from an optimal terminal decision at the current stage and the risk from an optimally designed future sampling scheme.	94
$\bar{\rho}_j\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}\right]$	The expectation of the risk over possible data z_j and external inputs w_j at stage j.	95
$\rho_j^*\left[z_{[j]}, w_{[j]}, d_{[j]}\right]$	The risk from an optimally-designed experiment on the system at stage j (taking into account all possible future samples).	95
d_j^*	Setting of the design parameter d_j which minimises the risk at stage j.	95

A.3.3 Approximate design algorithm

Symbol	Description	Introduced
$r_j^{(i)}\left[z_{[j]}, w_{[j]}, d_{[j]}\right]$	Function which approximates the risk $\rho_j[.]$ at wave i and stage j of the approximate backward induction procedure	101
$\bar{r}_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j]}\right]$	Function which approximates the expected risk $\bar{\rho}_j[.]$ at wave i of the approximation algorithm	101
$s_j^{(i)}\left[z_{[j-1]}, w_{[j-1]}, d_{[j-1]}\right]$	Approximation to the risk $\rho_j^*[.]$ at stage j and wave i of the approximation algorithm.	101
$\tilde{d}_j$	Candidate design generated from the approximate risk $\bar{r}_j^{(i)}[.]$ using algorithm 3.	119
$\mathcal{D}_j^{(i)}$	Candidate design space at wave i, stage j, characterised using algorithm 3.	119
$h_{jp}^{(i)}\left(z_{[j]}, w_{[j]}, d_{[j]}\right)$	Basis functions for the approximating risk emulator chosen at wave i of the approximation algorithm.	101
$\alpha_{jp}^{(i)}$	Regression coefficients for the j^{th} risk at the i^{th} wave of the approximation algorithm.	101
$u_j^{(i)}\left(z_{[j]}, w_{[j]}, d_{[j]}\right)$	Residual process of the emulator for the risk at the j^{th} at the i^{th} wave of the algorithm.	101
$R_{jk}^{(i)}$	Risk data point k generated to fit the risk emulator at stage j, wave i of the algorithm.	107
$\hat{d}_k$	Design that we actually select to perform the experiment at stage k (based on the output of the algorithm).	124
$v_k^{(i)}$	Expected value of perfect information (EVPI) for the design selected for the experiment at stage k (based on beliefs about the risk after wave i).	125

A.4 Design for developing models

A.4.1 Model structure

Symbol	Description	Introduced
n_s	Number of simulators in the reifying framework (including the reified simulator)	164
$f_i^{(k)}(\theta)$	i^{th} output from the simulator for the system at stage k of the development process, evaluated at input setting θ.	165
$g_j^{(k)}(\theta)$	j^{th} basis function for the k^{th} emulator in the reifying framework.	165
$\beta_{ij}^{(k)}$	ij^{th} basis coefficient for the k^{th} emulator in the reifying framework.	165
$r_i^{(k)}(\theta)$	i^{th} residual process for the k^{th} emulator in the reifying framework.	165
$\gamma_{ij}^{(k)}(\theta)$	Scaling function controlling the contribution from the j^{th} residual output at stage $(k-1)$ to the i^{th} residual output at stage k.	166
$\nu_i^{(k)}(\theta)$	Additional residual contribution to output i at the k^{th} development stage.	166
θ_j	j^{th} input at which we evaluate the simulator to obtain data for adjustment.	168
F_{ijk}	Output i from simulator k evaluated at input setting θ_j; superscript k has been lowered to clarify the adjustment equations.	167

A.4.2 Backward induction- evolving models

Symbol	Description	Introduced

$\rho_j^f\left[.,F_{[j]},\phi_{[j]}\right]$	Risk from the decision to stop sampling immediately after having constructed and run the j^{th} simulator, with decisions based on posterior beliefs represented by the posterior $p\left(q\middle\vert z_{[j-1]},w_{[j-1]},d_{[j-1]},F_{[j]},\phi_{[j]}\right)$.	178
$\psi_j\left[.,F_{[j]},\phi_{[j]}\right]$	Risk from choosing the optimal course of action between continuing and sampling z_j, or stopping and making an immediate decision based on the data $z_{[j-1]}$ and the simulator runs $F_{[j]}$.	177
$\bar{\psi}_j\left[.,F_{[j-1]},\phi_{[j]}\right]$	Expectation of ψ_j with respect to the distribution $p\left(F_j\middle\vert F_{[j-1]},\phi_{[j]}\right)$.	178
$\psi_j^*\left[.,F_{[j-1]},\phi_{[j-1]}\right]$	Risk from an optimal course of action at stage j, obtained by minimising over the design ϕ_j used for the simulator runs at this stage.	178
$c_j^f\left(\phi_j\right)$	Cost of building and running the simulator at stage j	178

A.4.3 Approximate backward induction- evolving models

Symbol	Description	Introduced
$t_j^{(i)}\left[.,F_{[j]},\phi_{[j]}\right]$	Emulator fitted as an approximation to $\psi_j\left[.\right]$ at the i^{th} wave of the approximate backward induction procedure.	184
$\bar{t}_j^{(i)}\left[.,F_{[j-1]},\phi_{[j]}\right]$	Emulator for the expected risk $\bar{\psi}_j\left[.\right]$ at the i^{th} wave of the approximate procedure, computed by integrating the emulator $t_j^{(i)}$ against our current beliefs about F_j.	184
$v_j^{(i)}\left[.,F_{[j-1]},\phi_{[j-1]}\right]$	Approximation to $\psi_j^*\left[.\right]$ at the i^{th} wave of the approximate procedure; characterised using the procedure in Section 5.5.5.	185
$T_{jk}^{(i)}$	Data used to fit the emulator $t_j^{(i)}$.	191

$\hat{h}_p^{(i)}(.)$	Basis functions for the emulator $t_j^{(i)}[.]$ fitted at the i^{th} wave of the procedure.	184
$\hat{\alpha}_{jp}^{(i)}$	Basis coefficients for the emulator $t_j^{(i)}[.]$ fitted at the i^{th} wave of the procedure.	184
$\hat{u}_j^{(i)}(.)$	Correlated residual component of the emulator $t_j^{(i)}[.]$ fitted at the i^{th} wave of the procedure.	184
$\hat{\xi}_j^{(i)}$	Nugget component of the emulator $t_j^{(i)}[.]$ fitted at the i^{th} wave of the procedure.	184

A.5 Bayes linear numerical modelling

A.5.1 Ordinary differential equations

ODE model

Symbol	Description	Introduced
$u_i(t,\xi)$	i^{th} component of the solution to the ODE at time t and parameter setting ξ (frequently abbreviated to $u_i(t)$).	227
ξ	Parameters governing the evolution of the ODE solution trajectory.	227
$u_i^{(0)}$	Initial condition for the i^{th} solution component ($u_i(t_0) = u_i^{(0)}$).	227
$f_i(u,t,\xi)$	Function specifying the behaviour of the time-derivative of the i^{th} solution component $\frac{d}{dt}(u_i(t))$.	227

Numerical schemes

Symbol	Description	Introduced
t_k	Time knots ($k = 0, 1, \ldots, n_t$) for the numerical solution.	228
$\hat{u}_i(t_k,\xi)$	Numerical solution at time knot t_k, parameter setting ξ (frequently)	228

$\phi_i(u, t_k, t_{k+1}, \xi)$	Numerical evolution function; generates numerical solution $\hat{u}(t_{k+1})$ from initial state $u(t_k)$ (at parameter setting ξ).	228
θ	Abbreviated notation for solver input set; $\theta = \{u, t_k, t_{k+1}, \xi\}$	242

Numerical discrepancy

Symbol	Description	Introduced
$\eta_i(u, t_k, t_{k+1}, \xi)$	Numerical discrepancy induced by numerically evolving from $u(t_k)$ to time-step t_{k+1} (commonly abbreviated $\eta(t_{k+1})$, with $u_i(t_{k+1}) = \hat{u}_i(t_{k+1}) + \eta_i(t_{k+1})$).	229
$g_p(\theta)$	p^{th} basis function for the regression component of the numerical discrepancy model.	242
β_{ip}	ip^{th} regression parameter for the numerical discrepancy model.	242
$r_i(\theta)$	i^{th} element of the residual component of the numerical discrepancy model.	242

System relationship

Symbol	Description	Introduced
$y_i(t)$	System value component i at time t.	232
ξ^*	Best input setting for the parameters ξ of the ODE model.	232
$\delta_i(t)$	Discrepancy between the ODE solution and the system (component i at time t).	232
z_{ik}	Data observed on component i the system at time t_k.	232
ϵ_{ik}	Measurement error on the observation z_{ik}.	232

A.5.2 Partial differential equations

PDE model

Symbol	Description	Introduced
$u_i(x,t,\xi)$	i^{th} component ($i=1,\ldots,n_u$) of the solution to the PDE at spatial location x and time t, for parameter setting ξ (generally abbreviated to $u(x,t)$).	265
$F_i(u,\ldots,t,\xi)$	Function which links the temporal partial derivative $\frac{\partial u_i}{\partial t}$ of the solution to the solution itself, its spatial partial derivatives $\{\frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}\}$ and the parameters ξ.	265
$f_i(x,t)$	Forcing function which drives the evolution of the i^{th} component.	265
$v_i(x)$	Function specifying the behaviour of the solution for $x \in T$ at the initial time $t = t_0$.	265
$w_i(x,t)$	Function specifying the behaviour of the solution on the boundary region $x \in \partial\Omega$ for all times t.	265

Finite element scheme

Symbol	Description	Introduced
$\hat{u}(x,t_k,\xi)$	Numerical approximation to the solution at time knot t_k, spatial location x and parameter setting ξ (generally abbreviated to $\hat{u}(x,t)$).	268
$\phi_i(x)$	i^{th} spatial basis function ($i=1,\ldots,n_\phi$) of the finite element approximation.	268
$\hat{u}_i(t_k)$	Coefficient of the i^{th} basis function in the finite element approximation.	268
θ	Abbreviation for the solver input set $\theta = \{\phi_1(x),\ldots,\phi_{n_\phi}(x), t_k, t_{k+1}, \xi^*\}$.	269
$K(\theta)$	Finite element solver matrix.	269
$A(\theta)$	Inverse of the finite element solver matrix ($A(\theta) = K(\theta)^{-1}$).	269

$b(\theta)$	Vector of boundary information.	269
$f(\theta)$	Vector of basis-integrated forcing values.	269

Numerical discrepancy

Symbol	Description	Introduced
$\eta(x, t_k)$	Discrepancy between the numerical solution and the true solution at time t_k.	271
$\eta^{(p)}(x, t_k)$	p^{th} partial derivative (wrt x) of the numerical discrepancy at time t_k.	271

Appendix B

Mean and covariance functions

In this chapter, we consider some of the mean and covariance functions that we might use in the construction of our emulators: in Section B.1, we consider the squared exponential covariance function, in Section B.2, we consider the Matèrn covariance function, and in Section B.3, we consider the use of splines as basis functions for an emulator. In all cases, we define the basis or covariance function, and we consider the derivatives or integrals of it that we may need to evaluate in order to be able to carry out the calculations in Sections 2.3.2, 2.4.1, 2.4.2 and 2.4.3.

B.1 Covariance: squared-exponential

The squared exponential covariance function is the most commonly used covariance function in this thesis. For a scalar function with inputs $\theta = (\theta_1, \ldots, \theta_{n_\theta})^{\mathrm{T}}$, the most general form is

$$\mathrm{Cov}\left[u\left(\theta\right), u\left(\theta'\right)\right] = k\left(\theta, \theta'\right) = v \exp\left[-\frac{1}{2} l\left(\theta, \theta'\right)^2\right]$$

where

$$l\left(\theta, \theta'\right) = \left[(\theta - \theta')^{\mathrm{T}} \Lambda (\theta - \theta')\right]^{1/2}$$

is the scaled distance between the points θ and θ', v is the marginal variance of the process, and Λ is a matrix whose eigenvalues and eigenvectors determine the magnitudes and directions of correlations in the process. A popular extension for

multi-output functions is

$$\mathrm{Cov}\left[u_i\left(\theta\right), u_j\left(\theta'\right)\right] = k_{ij}\left(\theta,\theta'\right) = v_{ij}\exp\left[-\frac{1}{2}l\left(\theta,\theta'\right)^2\right]$$

where v is a positive semi-definite matrix whose entries specify the marginal variances and covariances between function outputs at the same input location. In practice, it is often too difficult to confidently determine suitable values for a full matrix Λ, and use of a non-diagonal form also makes integration more difficult; therefore, Λ is frequently chosen to be diagonal.

B.1.1 Derivatives

As described in Section 2.3.2, for Gaussian or second-order prior belief specification, the relationship between a process and its partial derivatives can be easily derived: the derivative process is simply another process of the same type with mean and covariance functions which are correspondingly differentiated from the originals.

Once-differentiated First, we compute the covariance of $u\left(\theta\right)$ with its first partial derivative $\frac{\partial u}{\partial\theta_p}\left(\theta\right)$ with respect to a single input θ_p; as in Section 2.3.2, we have that

$$\begin{aligned}\mathrm{Cov}\left[\frac{\partial u_i}{\partial\theta_p}\left(\theta\right), u_j\left(\theta'\right)\right] &= \frac{\partial}{\partial\theta_p}k_{ij}\left(\theta,\theta'\right)\\ &= -d_p\left(\theta,\theta'\right)k_{ij}\left(\theta,\theta'\right)\end{aligned}$$

where

$$\begin{aligned}d_p\left(\theta,\theta'\right) &= \frac{1}{2}\frac{\partial}{\partial\theta_p}\left(l\left(\theta,\theta'\right)^2\right)\\ &= \sum_k \Lambda_{pk}(\theta_k-\theta'_k)\ .\end{aligned}$$

Twice-differentiated From the once-differentiated covariance, we can compute both the covariance between two first partial derivatives and the covariance between the function and its second partial derivative. The former is computed as

$$\begin{aligned}\mathrm{Cov}\left[\frac{\partial u_i}{\partial\theta_p}\left(\theta\right), \frac{\partial u_j}{\partial\theta_q}\left(\theta'\right)\right] &= \frac{\partial}{\partial\theta_p}\frac{\partial}{\partial\theta'_q}k_{ij}\left(\theta,\theta'\right)\\ &= \left[\Lambda_{pq} - d_p\left(\theta,\theta'\right)d_q\left(\theta,\theta'\right)\right]k_{ij}\left(\theta,\theta'\right)\end{aligned}$$

and the latter is

$$\begin{aligned}\mathrm{Cov}\left[\frac{\partial^2 u_i}{\partial\theta_p\partial\theta_q}(\theta), u_j(\theta')\right] &= \frac{\partial}{\partial\theta_p}\frac{\partial}{\partial\theta_q}k_{ij}(\theta,\theta') \\ &= \left[d_p(\theta,\theta')\,d_q(\theta,\theta') - \Lambda_{pq}\right]k_{ij}(\theta,\theta') \\ &= -\mathrm{Cov}\left[\frac{\partial u_i}{\partial\theta_p}(\theta), \frac{\partial u_j}{\partial\theta_q}(\theta')\right] .\end{aligned}$$

Higher-order derivatives We can compute higher-order derivatives of the covariance function in the same way as above.

B.1.2 Integrals

In order to translate beliefs about a function into beliefs about its integral (see Sections 2.3.2 and 2.4.1), or to propagate uncertainty on a subset of inputs through an uncertain function (see Sections 2.4.2 and 2.4.3), we must also be able to compute the integrals of certain products of the basis and covariance functions of an emulator with respect to particular probability distributions. We consider both of these cases for the case of a squared exponential covariance function.

Beliefs about the integral In the first case, we want to compute the covariance of $u(\theta)$ with $\bar{u}(b)$, the integral of the function with respect to $p(a)$

$$\bar{u}_i(b) = \int u_i(a,b)\,p(a)\,da .$$

As discussed in Section 2.3.2, this is simply the integral of the covariance function $k(.,.)$ in its first argument

$$\begin{aligned}\mathrm{Cov}\left[\bar{u}_i(b), u_j(\theta')\right] &= \bar{k}_{ij}(b,\theta') \\ &= \int k_{ij}(a,b,\theta')\,p(a)\,da .\end{aligned}$$

The covariance between $\bar{u}(b)$ and $\bar{u}(b')$ can then be found by integrating a second time

$$\begin{aligned}\mathrm{Cov}\left[\bar{u}_i(b), \bar{u}_j(b')\right] &= \bar{\bar{k}}_{ij}(b,b') \\ &= \iint k_{ij}(a,b,a',b')\,p(a)\,p(a')\,da\,da' .\end{aligned}$$

For the squared exponential covariance, we compute these integrals in two cases; the case where the a have a joint multivariate Gaussian distribution, and the case where they are independently uniformly distributed.

Gaussian case First, the multivariate Gaussian case, where

$$p\left(a\right) = \frac{1}{(2\pi)^{n_a/2}|V_a|^{1/2}} \exp\Big[-\frac{1}{2}(a-\mu_a)^{\mathrm{T}}V_a^{-1}(a-\mu_a)\Big] .$$

To simplify the calculation, we assume that the covariance function is separable in the parameters a and b, so

$$k_{ij}\left(\theta,\theta'\right) = v_{ij}c_a\left(a,a'\right)c_b\left(b,b\right)$$

where $c_a\left(.,.\right)$ and $c_b\left(.,.\right)$ are squared exponential correlation functions with correlation matrices Λ_a and Λ_b. The integrated covariance is then

$$\bar{k}_{ij}\left(b,\theta'\right) = v_{ij}\bar{c}_a\left(a'\right)c_b\left(b,b\right)$$

where

$$\begin{aligned}\bar{c}_a\left(a'\right) &= \int \exp\Big[-\frac{1}{2}(a-a')^{\mathrm{T}}\Lambda_a(a-a')\Big]p\left(a\right)da \\ &= \left(\frac{|\hat{\Lambda}_a|}{|\Lambda_a|}\right)^{1/2}\exp\Big[-\frac{1}{2}(\mu_a-a')^{\mathrm{T}}\hat{\Lambda}_a(\mu_a-a')\Big]\end{aligned}$$

and

$$\hat{\Lambda}_a = \Big[V_a+\Lambda_a^{-1}\Big]^{-1} .$$

If we integrate for a second time, we find that

$$\bar{\bar{k}}_{ij}\left(b,b'\right) = v_{ij}\bar{\bar{c}}_a c_b\left(b,b\right)$$

where

$$\begin{aligned}\bar{\bar{c}}_a &= \iint \exp\Big[-\frac{1}{2}(a-a')^{\mathrm{T}}\Lambda_a(a-a')\Big]p\left(a\right)p\left(a'\right)dada' \\ &= \left(\frac{|\hat{\hat{\Lambda}}_a|}{|\Lambda_a|}\right)^{1/2}\exp\Big[-\frac{1}{2}(\mu_a-\mu_a')^{\mathrm{T}}\hat{\hat{\Lambda}}_a(\mu_a-\mu_a')\Big]\end{aligned}$$

and

$$\hat{\hat{\Lambda}}_a = \Big[V_a+V_a'+\Lambda_a^{-1}\Big]^{-1} .$$

Uniform case Second, the case where each of the elements $\{a_q\}$ has an independent uniform distribution, i.e.

$$p(a) = \prod_{q=1}^{n_a} p(a_q) \qquad\qquad p(a_q) = \frac{1}{u_{a_q} - l_{a_q}} .$$

In this instance, we assume that the covariance function is separable in each of the elements of a, so that

$$k_{ij}(\theta, \theta') = v_{ij}\Big[\prod_{q=1}^{n_a} c_{a_q}\left(a_q, a_q'\right)\Big] c_b(b, b')$$

where

$$c_{a_q}\left(a_q, a_q'\right) = \exp\Big[-\frac{\lambda_{a_q}}{2}(a_q - a_q')^2\Big]$$

is a squared exponential correlation function for a scalar input. Integrating this once, we find that

$$\bar{k}_{ij}(b, \theta') = v_{ij}\Big[\prod_{q=1}^{n_a} \bar{c}_{a_q}\left(a_q'\right)\Big] c_b(b, b')$$

where the individual integrated terms are

$$\begin{aligned}
\bar{c}_{a_q}\left(a_q'\right) &= \int c_{a_q}\left(a_q, a_q'\right) p(a_q)\, da_q \\
&= \frac{1}{u_{a_q} - l_{a_q}} \int_{l_{a_q}}^{u_{a_q}} \exp\Big[-\frac{\lambda_{a_q}}{2}(a_q - a_q)^2\Big] da_q \\
&= \frac{1}{u_{a_q} - l_{a_q}} \sqrt{\frac{2\pi}{\lambda_{a_q}}} \Big[\Phi\left(\sqrt{\lambda_{a_q}}(u_{a_q} - a_q')\right) - \Phi\left(\sqrt{\lambda_{a_q}}(l_{a_q} - a_q')\right)\Big]
\end{aligned}$$

where $\Phi(.)$ is the CDF of the standard univariate Gaussian distribution. Integrating for a second time, we have

$$\bar{\bar{k}}_{ij}(b, b') = v_{ij}\Big[\prod_{q=1}^{n_a} \bar{\bar{c}}_{a_q}\Big] c_b(b, b')$$

where

$$\bar{\bar{c}}_{a_q} = \iint c_{a_q}\left(a_q, a_q'\right) p(a_q)\, p\left(a_q'\right) da_q da_q' .$$

To compute this quantity, we use the fact that the indefinite integral of a normal CDF is [Owen, 1980]

$$\begin{aligned}
\psi(s, v, w) &= \int \Phi(s(v - w))\, dw \\
&= \frac{1}{s}\big(s(v - w)\Phi(s(v - w)) + \mathcal{N}(s(v - w))\big)
\end{aligned}$$

where $\mathcal{N}(.)$ is the density function for the standard univariate Gaussian. The twice-integrated covariance function is then

$$\bar{\bar{c}}_{a_q} = \frac{1}{(u_{a_q} - l_{a_q})(u_{a'_q} - l_{a'_q})} \sqrt{\frac{2\pi}{\lambda_{a_q}}} \Big[\psi\left(\sqrt{\lambda_{a_q}}, u_{a_q}, u_{a'_q}\right) - \psi\left(\sqrt{\lambda_{a_q}}, u_{a_q}, l_{a'_q}\right) - \psi\left(\sqrt{\lambda_{a_q}}, l_{a_q}, u_{a'_q}\right) + \psi\left(\sqrt{\lambda_{a_q}}, l_{a_q}, l_{a'_q}\right) \Big] .$$

Uncertainty propagation In order to propagate uncertainty through an emulator, in addition to the integrals evaluated in the previous paragraph, we must be able to evaluate integrals of the following type (Section 2.4.2)

$$h_{k,l}(b, b') = \int_{\mathcal{A}} g_k(a^*, b)\, g_l(a^*, b')\, p(a^*)\, da^* \tag{B.1.1}$$

$$v_{k,rsj}(b, b') = \int_{\mathcal{A}} g_k(a^*, b)\, k_{rj}(\theta_s, a^*, b')\, p(a^*)\, da^* \tag{B.1.2}$$

$$w_{ikl,jpq}(b, b') = \int_{\mathcal{A}} k_{ik}(a^*, b, \theta_l)\, k_{jp}(a^*, b', \theta_q)\, p(a^*)\, da^* . \tag{B.1.3}$$

We evaluate each of these in the case where the basis $g(.)$ consists of products of polynomial functions of the inputs, the covariance function $k(.,.)$ is a squared exponential which is separable in each input component, and the probability distribution $p(a^*) = \prod_{q=1}^{n_a} p\left(a_q^*\right)$, where $p\left(a_q^*\right)$ is a uniform distribution.

For polynomial basis functions $g_k(\theta)$ and a product of uniform distributions, it is easy to obtain expressions for the $h_{k,l}(b, b')$; we must simply evaluate a polynomial integral at the limits of the distributions. For integrals of the second type ((B.1.2)), if the covariance function factorises as

$$k_{ij}(\theta, \theta') = v_{ij} \Big[\prod_{q=1}^{n_a} c\left(a_q, a'_q\right) \Big] c(b, b)$$

then we can split the integral as follows

$$v_{k,rsj}(b, b') = v_{rj} c\left(b^{(s)}, b'\right) \int_{\mathcal{A}} g_k(a^*, b) \Big[\prod_{q=1}^{n_a} c\left(a_q^{(s)}, a_q^*\right) \Big] p(a^*)\, da^* .$$

All of the integrals that we must perform, then, are of the following form

$$\int_{\mathcal{A}} (a')^{\alpha} c(a, a')\, p(a')\, da' \tag{B.1.4}$$

for some scalar power α (where we have dropped the subscript q and the superscript $*$, since what follows holds for any scalar component of the full parameter set). We cannot evaluate these integrals directly, but we can instead evaluate integrals of the form

$$\int \big(\lambda(a-a')\big)^{\alpha} \exp\Big[-\frac{\lambda}{2}(a-a')^2\Big] da' \tag{B.1.5}$$

for all powers $1, 2, \ldots, \alpha$, and then compute (B.1.4) by expanding out the polynomial and substituting. First, in the case $\alpha = 0$, the indefinite integral (B.1.4) is

$$\int \exp\Big[-\frac{\lambda}{2}(a-a')^2\Big] da' = \sqrt{\frac{2\pi}{\lambda}}\Phi\Big(\sqrt{\lambda}(a'-a)\Big)$$

and in the case $\alpha = 1$, integrating and rearranging (B.1.5) gives

$$\begin{aligned}\int a' \exp\Big[-\frac{\lambda}{2}(a-a')^2\Big] da' \\ &= a\int \exp\Big[-\frac{\lambda}{2}(a-a')^2\Big] da' - \frac{1}{\lambda}\exp\Big[-\frac{\lambda}{2}(a-a')^2\Big] \\ &= a\sqrt{\frac{2\pi}{\lambda}}\Phi\Big(\sqrt{\lambda}(a'-a)\Big) - \frac{1}{\lambda}\exp\Big[-\frac{\lambda}{2}(a-a')^2\Big].\end{aligned}$$

For general values of α, integrating (B.1.5) by parts yields

$$\begin{aligned}\int \big(\lambda(a-a')\big)^{\alpha} \exp\Big[-\frac{\lambda}{2}(a-a')^2\Big] da' \\ &= \big(\lambda(a-a')\big)^{\alpha-1} \exp\Big[-\frac{\lambda}{2}(a-a')^2\Big] \\ &\quad + \lambda(\alpha-1)\int \big(\lambda(a-a')\big)^{\alpha-2} \exp\Big[-\frac{\lambda}{2}(a-a')^2\Big] da'\end{aligned}$$

and continuing to integrate in this manner eventually gives [Owen, 1980]

$$\begin{aligned}\int \big(\lambda(a-a')\big)^{\alpha} \exp\Big[-\frac{\lambda}{2}(a-a')^2\Big] da' \\ &= \begin{cases} c\,(a,a')\,\sum_{\beta=0}^{n} \lambda^{n-\beta}\frac{(2n)!!}{(2\beta)!!}\big(\lambda(a-a')\big)^{2\beta} & \text{for odd } \alpha = 2n+1 \\ c\,(a,a')\,\sum_{\beta=0}^{n} \lambda^{n-\beta}\frac{(2n+1)!!}{(2\beta+1)!!}\big(\lambda(a-a')\big)^{2\beta+1} & \\ \quad + \lambda^n(2n+1)!!\sqrt{\frac{2\pi}{\lambda}}\Phi\big(\sqrt{\lambda}(a'-a)\big) & \text{for even } \alpha = 2n+2 \end{cases}\end{aligned}$$

where $n!!$ is the double factorial of n (equal to the product of all even numbers from 2 to n for even n, and the product of all odd numbers from 1 to n for odd n). Expressions for the integral (B.1.4) can now be computed by evaluating the above

expression for all powers $1, \ldots, \alpha$, expanding the polynomial expression in the integrand, and then back substituting the results for all lower powers.

Lastly, we must evaluate integrals of two covariance functions against each other (of the form (B.1.3), again with separable squared exponential covariance functions and uniform distributions). Under these assumptions, the integral factorises as

$$w_{ikl,jpq}\left(b, b'\right) = v_{ik} v_{jp} c\left(b, b^{(l)}\right) c\left(b', b^{(q)}\right) \int_{\mathcal{A}} \Big[\prod_{s=1}^{n_a} c\left(a_s^*, a_s^{(l)}\right) c\left(a_s^*, a_s^{(q)}\right) \Big] p\left(a^*\right) da^* \ . \tag{B.1.6}$$

Under the assumption of independent uniform distributions, in order to evaluate (B.1.6), we must compute a series of integrals of the following form

$$I(a', a'') = \frac{1}{(u-l)} \int_l^u \exp\Big[-\frac{\lambda}{2}(a-a')^2 \Big] \exp\Big[-\frac{\lambda}{2}(a-a'')^2 \Big] da \ .$$

Expanding out and completing the square in the exponent, we find that

$$\begin{aligned} I(a', a'') = \frac{1}{(u-l)} \int_l^u \exp\Big[-\frac{2\lambda}{2}\Big(a^2 - 2\frac{(a'+a'')}{2} a + \Big(\frac{a'+a''}{2} \Big)^2 \Big) \Big] da \\ \times \exp\Big[-\frac{\lambda}{2}\Big((a')^2 - \frac{(a'+a'')^2}{2} + (a'')^2 \Big) \Big] \ . \end{aligned}$$

Factorising the exponents and performing the integral then gives

$$\begin{aligned} I(a', a'') = \frac{1}{(u-l)} \sqrt{\frac{\pi}{\lambda}} \Big[\Phi\left(\sqrt{2\lambda}(u - \frac{a'+a''}{2}) \right) - \Phi\left(\sqrt{2\lambda}(l - \frac{a'+a''}{2}) \right) \Big] \\ \times \exp\Big[-\frac{\lambda/2}{2}(a'-a'')^2 \Big] \ . \end{aligned}$$

B.2 Covariance: Matèrn

The Matèrn covariance function is a much more flexible covariance function, which allows us to control the smoothness of the process as well as its correlation structure. It has the following form

$$\mathrm{Cov}\left[u\left(\theta\right), u\left(\theta'\right) \right] = k\left(l\left(\theta, \theta'\right), \nu \right) = \frac{1}{2^{\nu-1}\Gamma\left(\nu\right)} \left(l\left(\theta, \theta'\right) \right)^{\nu} K_{\nu}\left(l\left(\theta, \theta'\right) \right)$$

where ν is a differentiability parameter, $\Gamma(.)$ is the Gamma function and $K_\nu(.)$ is a modified Bessel function of the second kind. As in the squared exponential case, we can define covariance functions for multi-output functions as

$$\begin{aligned}\mathrm{Cov}\left[u_i(\theta), u_j(\theta')\right] &= k_{ij}\left(l(\theta,\theta'),\nu\right)\\ &= \frac{v_{ij}}{2^{\nu-1}\Gamma(\nu)}(l(\theta,\theta'))^{\nu}K_\nu\left(l(\theta,\theta')\right) .\end{aligned}$$

In what follows, we consider only derivatives of the single-output covariance function, though the results are easily extended to the multi-output case.

B.2.1 Derivatives

We use the following property of the Bessel function to find derivatives of the Matèrn covariance [Wolfram, 2017]

$$\frac{\partial}{\partial z}K_\nu(z) = -\left[\frac{\nu}{z}K_\nu(z) + K_{\nu-1}(z)\right] .$$

Once-differentiated We begin by applying the chain rule

$$\begin{aligned}\mathrm{Cov}\left[\frac{\partial u}{\partial\theta_p}(\theta), u(\theta')\right] &= \frac{\partial}{\partial\theta_p}k\left(l(\theta,\theta'),\nu\right)\\ &= \left[\frac{\partial}{\partial l}k(l)\right].\frac{\partial l}{\partial\theta_p} .\end{aligned}$$

The derivative of the covariance is

$$\begin{aligned}\frac{\partial}{\partial l}k(l) = &\quad \frac{1}{2^{\nu-1}\Gamma(\nu)}\nu l^{\nu-1}K_\nu(l)\\ &- \frac{1}{2^{\nu-1}\Gamma(\nu)}l^{\nu}\left[\frac{\nu}{l}K_\nu(l) + K_{\nu-1}(l)\right]\\ = &\quad -\frac{1}{2^{\nu-1}\Gamma(\nu)}l^{\nu}K_{\nu-1}(l)\end{aligned}$$

and the derivative of the distance function is

$$\frac{\partial}{\partial\theta_p}[l(\theta,\theta')] = \frac{1}{l(\theta,\theta')}\left[\sum_i \Lambda_{pi}(\theta_i-\theta'_i)\right]$$

which gives

$$\frac{\partial}{\partial\theta_p}k\left(l(\theta,\theta')\right) = -\frac{\left[d_p(\theta,\theta')\right]}{2(\nu-1)}k\left(l(\theta,\theta'),\nu-1\right)$$

where $d_p(\theta,\theta')$ is defined as in the squared-exponential case.

Twice-differentiated If we apply the same procedure again, we obtain the following as the covariance between the partial derivatives with respect to two different variables at inputs θ and θ'

$$\begin{aligned}\mathrm{Cov}\left[\frac{\partial u}{\partial \theta_p}(\theta), \frac{\partial u}{\partial \theta_q}(\theta')\right] &= \frac{\partial}{\partial \theta'_q}\frac{\partial}{\partial \theta_p}k\left(l\left(\theta,\theta'\right),\nu\right) \\ &= \frac{1}{2(\nu-1)}\Big[\Lambda_{pq}k\left(l\left(\theta,\theta'\right),\nu-1\right) \\ &\qquad - \frac{d_p\left(\theta,\theta'\right)d_q\left(\theta,\theta'\right)}{2(\nu-2)}k\left(l\left(\theta,\theta'\right),\nu-2\right)\Big].\end{aligned}$$

If both of the derivatives are taken with respect to the un-primed variable, then we obtain the covariance between the second partial derivative and the original process

$$\begin{aligned}\mathrm{Cov}\left[\frac{\partial^2 u}{\partial \theta_p \partial \theta_q}(\theta), u(\theta')\right] &= \frac{1}{2(\nu-1)}\Big[-\Lambda_{pq}k\left(l\left(\theta,\theta'\right),\nu-1\right) \\ &\qquad + \frac{d_p\left(\theta,\theta'\right)d_q\left(\theta,\theta'\right)}{2(\nu-2)}k\left(l\left(\theta,\theta'\right),\nu-2\right)\Big].\end{aligned}$$

B.2.2 Integrals

Unfortunately, there is no indefinite integral of the plain Matèrn covariance function available in the literature- this is unfortunate, because this covariance is more appropriate in a variety of situations where we wish to be able to control the differentiability of the function which we are modelling. An alternative strategy is available, however; if we know that we want to integrate with respect to input θ_k over a particular range, and we are happy to make a uniform distributional assumption for θ_k over this range, then we could use the second partial derivative obtained in Section B.2.1 as the covariance function for the original process, and then 'reverse the differentiation' in order to obtain the covariance of the integral.

B.3 Mean: Splines

Splines are a popular smoothing tool in the literature- see, for example, de Boor [1986], Vermeulen et al. [1992]- while many different types of spline function exist, we only consider B-splines (basic splines), which are perhaps the most commonly used. For functions of a single input variable, splines are built recursively as follows: we

choose a non-decreasing sequence of knots $\{t_i\}$, $i = 1, \ldots, n_t$ which spans the input space, and define the first-order B-splines as indicator functions for the intervals between knots

$$b_{i1}(\theta) = \begin{cases} 1 & \text{, if } t_i \leq \theta < t_{i+1} \\ 0 & \text{, else.} \end{cases}$$

From these first-order splines, we obtain all higher order splines by recursion: using k to index order, and defining

$$\omega_{ik}(\theta) = \frac{\theta - t_i}{t_{i+k-1} - t_i}$$

we have

$$b_{ik}(\theta) = \omega_{ik}(\theta) b_{i(k-1)}(\theta) + (1 - \omega_{ik}(\theta)) b_{(i+1)(k-1)}(\theta) \ . \tag{B.3.7}$$

It is easy to verify (see de Boor [1986]) that each $b_{ik}(\theta)$ defined in this way is a compactly-supported polynomial function of degree k, which vanishes outside the interval $[t_i, t_{i+k})$. Using $b_k(\theta) = (b_{1k}(\theta), \ldots, b_{n_b k}(\theta))^{\mathrm{T}}$ to denote the vector of all spline functions of a given degree, we simply use the B-splines as our regression basis; for a scalar function $f(.)$ of scalar input θ, our representation is

$$f(\theta) = \sum_{i=1}^{n_t} \beta_i b_{ik}(\theta) \tag{B.3.8}$$

where β is, as before, a coefficient governing the magnitude of the contribution from each basis function.

B.3.1 Derivatives

Using the representation (B.3.8), the derivative of the function is simply the weighted sum of the basis function derivatives

$$\frac{d}{d\theta}\big(f(\theta)\big) = \sum_{i=1}^{n_b} \beta_i \frac{d}{d\theta}\big(b_{ik}(\theta)\big) \ .$$

Vermeulen et al. [1992] provide a more convenient representation of this derivative as a weighted sum of spline functions of one order lower

$$\frac{d}{d\theta}\big(f(\theta)\big) = \sum_{i=1}^{n_b} \beta_i^{(1)} b_{i(k-1)}(\theta)$$

where the coefficients are

$$\beta_i^{(1)} = \frac{k}{t_{i+k} - t_i}\beta_i - \frac{k}{t_{i+k-1} - t_{i-1}}\beta_{i-1}$$

and we define $\beta_{-1}^{(1)} = \beta_{n_t+1}^{(1)} = 0$.

B.3.2 Integrals

Similarly, the indefinite integral of the basis representation (B.3.8) is

$$\int f(\theta)\, d\theta = \sum_{i=1}^{n_b} \beta_i \Big[\int b_{ik}(\theta)\, d\theta \Big] .$$

Again, Vermeulen et al. [1992] derive a simple representation for right-hand side in terms of spline functions of one order higher

$$\int f(\theta)\, d\theta = \sum_{i=1}^{n_b} \beta_i^{(-1)} b_{i(k+1)}(\theta)$$

where the coefficients are computed in terms of the β as follows

$$\beta_i^{(-1)} = \frac{t_{i+k+1} - t_i}{k+1} \Big[\frac{k+1}{t_{i+k} - t_{i-1}} \beta_{i-1}^{(-1)} + \beta_i \Big]$$

where we define $\beta_{-1}^{(-1)} = 0$. To evaluate this integral, we need to introduce an additional knot t_{n_t+1}; this is arbitrary, provided it satisfies $t_{n_t+1} \geq t_{n_t}$. Choosing that $t_{n_t+1} = t_{n_t}$ can help to simplify implementation.

Appendix C

Diffusion equation example-implementation details

In this appendix, we provide implementation details for the diffusion equation example presented in Section 6.3.4. In Section B.1, we detail a procedure for evaluating the real solution of the equation on an infinite domain, for particular forms of the initial condition and forcing functions. Then, in Section B.2, we carry out the basis function integrals required for the finite element implementation in this case.

C.1 Evaluating the real solution

If we solve (6.3.22) on the spatial domain $\Omega = (-\infty, \infty)$, and assume that the initial condition (at $t = 0$) is a Dirac delta function at the origin, then the solution for all $t \in (0, \infty)$ is a Gaussian function with variance parameter $2\kappa t$

$$\Phi\left(x, t\right) = \frac{1}{\sqrt{4\pi\kappa t}} \exp\Big[-\frac{x^2}{4\kappa t} \Big] .$$

Φ is known as the fundamental solution of the diffusion equation; we can obtain solutions on this domain for other initial condition and forcing specifications by convolving these with the fundamental solution [Cannon, 1985]. Clearly, if we make corresponding Gaussian specifications for our initial condition and forcing functions, then this convolution will result in a relatively simple form for the solution.

If we fix $f(x,t) = 0$ and choose

$$v(x) = \frac{1}{\sigma_v\sqrt{2\pi}} \exp\Big[-\frac{1}{2{\sigma_v}^2}(x-\mu_v)^2\Big]$$

then we can find the solution to the resulting initial value problem as

$$\begin{aligned} u_v(x,t) &= \int_\Omega \Phi(x-\xi,t)\, v(\xi)\, d\xi \\ &= \frac{1}{\sqrt{2\pi(2\kappa t+{\sigma_v}^2)}} \exp\Big[-\frac{1}{2}\frac{(x-\mu_v)^2}{(2\kappa t+{\sigma_v}^2)}\Big] \ . \end{aligned} \tag{C.1.1}$$

Alternatively, if we assume that $v(x) = 0$ and choose a Gaussian form for $f(x,t)$

$$f(x,t) = f(x) = \frac{1}{\sigma_f\sqrt{2\pi}} \exp\Big[-\frac{1}{2{\sigma_f}^2}(x-\mu_f)^2\Big]$$

then the solution that we obtain is

$$\begin{aligned} u_f(x,t) &= \int_0^t \int_\Omega \Phi(x-\xi,t-\tau)\, f(\xi,\tau)\, d\xi d\tau \\ &= \int_0^t \frac{1}{\sqrt{2\pi(2\kappa(t-\tau)+{\sigma_f}^2)}} \exp\Big[-\frac{1}{2}\frac{(x-\mu_f)^2}{(2\kappa(t-\tau)+{\sigma_f}^2)}\Big] d\tau \ . \end{aligned} \tag{C.1.2}$$

Additionally, because of the linearity of the equation (6.3.22), we can find the forced solution which also satisfies the initial value problem by assuming that the solution is

$$u(x,t) = u_v(x,t) + u_f(x,t)$$

where u_v ((C.1.1)) satisfies the un-forced initial value problem, and u_f ((C.1.2)) satisfies the forced problem for a zero initial condition. This solution, which is completely determined through specification of the location parameters $\{\mu_v, \mu_f\}$ and scale parameters $\{\sigma_v, \sigma_f\}$ for the initial and forcing functions is the one that we study in the example of Section 6.3.4. To evaluate the integral in (C.1.2), we use the 'integral' numerical integration function in Matlab.

C.2 Finite element: basis integrals

In this section, we provide details of the basis function calculations that must be performed in order to implement the finite element scheme for the diffusion equation

example. The basis functions for this example are defined in equation (6.3.23); in order to evaluate D and K, we must compute three different kinds of integral

$$I^{(p)}_{A_{ij}} = \int_\Omega \phi_i(x)\,(x-x_j)^p \phi_j(x)\,dx \tag{C.2.3}$$

$$I^{(p)}_{B_{ij}} = \int_\Omega \frac{\partial}{\partial x}\big(\phi_i(x)\big).p(x-x_j)^{p-1}\phi_j(x)\,dx \tag{C.2.4}$$

$$I^{(p)}_{C_{ij}} = \int_\Omega \frac{\partial}{\partial x}\big(\phi_i(x)\big).(x-x_j)^p \frac{\partial}{\partial x}\big(\phi_j(x)\big)dx \tag{C.2.5}$$

for $p = 0, 1, 2$. Since the basis function ϕ_i is only non-zero on the interval $x_{i-1} \leq x < x_{i+1}$, all of the above integrals are only non-zero for $j = (i-1)$, $j = i$ and $j = (i+1)$. The derivative of the basis function (6.3.23) is

$$\begin{cases} \frac{1}{d_i} & \text{for } x_{i-1} \leq x < x_i \\ -\frac{1}{d_{i+1}} & \text{for } x_i \leq x < x_{i+1} \\ 0 & \text{else.} \end{cases}$$

For each of the integral types (C.2.3), (C.2.4), (C.2.5), we outline the computation of the integral for the central case $i = j$, and we tabulate the values of the other cases.

First case We first consider the integrals of type (C.2.3). For the central case $i = j$, the integral has the following form

$$\begin{aligned} I^{(p)}_{A_{ij}} = &\int_{x_{i-1}}^{x_i} \frac{(x-x_{i-1})^2}{d_i^2}(x-x_i)^p dx \\ &+ \int_{x_{i-1}}^{x_i} \frac{(x_{i+1}-x)^2}{d_{i+1}^2}(x-x_i)^p dx\ . \end{aligned} \tag{C.2.6}$$

Consider the first term; integrating once by parts gives

$$\begin{aligned} \int_{x_{i-1}}^{x_i} \frac{(x-x_{i-1})^2}{d_i^2}(x-x_i)^p dx = &\frac{1}{d_i^2(p+1)}\Big[(x-x_{i-1})^2(x-x_i)^{p+1}\Big]_{x_{i-1}}^{x_i} \\ &- \frac{2}{d_i^2(p+1)}\int_{x_{i-1}}^{x_i}(x-x_{i-1})(x-x_i)^{p+1}dx \end{aligned}$$

where we see that the first term vanishes. Integrating the second term by parts once more gives

$$\begin{aligned}\int_{x_{i-1}}^{x_i} \frac{(x-x_{i-1})^2}{d_i^2}(x-x_i)^p dx &= \frac{2}{d_i^2(p+1)(p+2)} \int_{x_{i-1}}^{x_i} (x-x_i)^{p+2} dx \\ &= \frac{2}{d_i^2(p+1)(p+2)(p+3)} \Big[(x-x_i)^{p+3}\Big]_{x_{i-1}}^{x_i} \\ &= \frac{2}{(p+1)(p+2)(p+3)}(-1)^p (d_i)^{p+1}\end{aligned}$$

which is negative for odd powers p and positive for even ones. We can now evaluate the second (right-hand) component of (C.2.6) by noting that each derivative of the first term will cause another factor of -1 to appear; differentiating this term twice causes these factors to cancel, giving

$$\begin{aligned}\int_{x_{i-1}}^{x_i} \frac{(x_{i+1}-x)^2}{d_{i+1}^2}(x-x_i)^p dx &= \frac{2}{d_{i+1}^2(p+1)(p+2)(p+3)} \Big[(x-x_i)^{p+3}\Big]_{x_i}^{x_{i+1}} \\ &= \frac{2}{(p+1)(p+2)(p+3)}(d_{i+1})^{p+1} \ .\end{aligned}$$

Following a similar procedure for $j=(i-1)$, we obtain

$$\begin{aligned}\int_{x_{i-1}}^{x_i} \frac{(x-x_{i-1})}{d_i}(x-x_{i-1})^p \frac{(x_i-x)}{d_i} dx &= \frac{1}{d_i^2} \int_{x_{i-1}}^{x_i} (x-x_{i-1})^{p+1}(x_i-x) dx \\ &= \frac{1}{d_i^2(p+2)} \int_{x_{i-1}}^{x_i} (x-x_{i-1})^{p+2} dx \\ &= \frac{1}{d_i^2(p+2)(p+3)} \Big[(x-x_{i-1})^{p+3}\Big]_{x_{i-1}}^{x_i} \\ &= \frac{(d_i)^{p+1}}{(p+2)(p+3)}\end{aligned}$$

which is $(p+1)/2$ times the right-hand integral from the central term. Following the same procedure for $j=(i+1)$ gives

$$\int_{x_i}^{x_{i+1}} \frac{(x_{i+1}-x)}{d_{i+1}}(x-x_{i+1})^p \frac{(x-x_i)}{d_{i+1}} dx = \frac{(-1)^p (d_{i+1})^{p+1}}{(p+2)(p+3)}$$

which is $(p+1)/2$ times the left-hand integral from the central term. These results are evaluated for all powers $p=0,1,2$ in table C.1.

$I^{(p)}_{A_{ij}}$	$j=(i-1)$	$j=i$	$j=(i+1)$
$p=0$	$\frac{d_i}{6}$	$\frac{d_i}{3}+\frac{d_{i+1}}{3}$	$\frac{d_{i+1}}{6}$
$p=1$	$\frac{d_i^2}{12}$	$-\frac{d_i^2}{12}+\frac{d_{i+1}^2}{12}$	$-\frac{d_{i+1}^2}{12}$
$p=2$	$\frac{d_i^3}{20}$	$\frac{d_i^3}{30}+\frac{d_{i+1}^3}{30}$	$\frac{d_{i+1}^3}{20}$

Table C.1: Values of the basis function integrals of type (C.2.3) for all powers p of the linear term and all elements j for which the integral is non-zero.

Second case For the central case ($i=j$) of the second type of integral (equation (C.2.4)), for $p >= 2$ we must evaluate

$$I^{(p)}_{B_{ii}} = \int_{x_{i-1}}^{x_i} \frac{1}{d_i} p(x-x_i)^{p-1} \frac{(x-x_{i-1})}{d_i} dx + \int_{x_i}^{x_{i+1}} -\frac{1}{d_{i+1}} p(x-x_i)^{p-1} \frac{(x_{i+1}-x)}{d_{i+1}} dx \ .$$

Considering the first term, we integrate by parts to find that

$$\begin{aligned}\int_{x_{i-1}}^{x_i} \frac{1}{d_i} p(x-x_i)^{p-1} \frac{(x-x_{i-1})}{d_i} dx &= -\frac{1}{d_i^2} \int_{x_{i-1}}^{x_i} (x-x_i)^p dx \\ &= -\frac{1}{d_i^2} \frac{1}{(p+1)} \Big[(x-x_i)^{p+1} \Big]_{x_{i-1}}^{x_i} \\ &= \frac{(-1)^p}{(p+1)} (d_i)^{p-1} \ .\end{aligned}$$

Using the same technique, we evaluate the second term as

$$\int_{x_i}^{x_{i+1}} -\frac{1}{d_{i+1}} p(x-x_i)^{p-1} \frac{(x_{i+1}-x)}{d_{i+1}} = \frac{-1}{(p+1)} (d_{i+1})^{p-1} \ .$$

The off-diagonal terms are as follows: for $j=(i-1)$, we have that

$$\begin{aligned} I_{B^{(p)}_{i(i-1)}} &= \int_{x_{i-1}}^{x_i} \frac{1}{d_i} p(x-x_{i-1})^{p-1} \frac{(x_i-x)}{d_i} dx \\ &= \frac{1}{d_i^2} \int_{x_{i-1}}^{x_i} (x-x_{i-1})^p dx \\ &= \frac{1}{(p+1)} (d_i)^{p-1} \end{aligned}$$

$I^{(p)}_{B_{ij}}$	$j=(i-1)$	$j=i$	$j=(i+1)$
$p=1$	$\frac{1}{2}$	0	$-\frac{1}{2}$
$p=2$	$\frac{d_i}{3}$	$\frac{d_i}{3}-\frac{d_{i+1}}{3}$	$-\frac{d_{i+1}}{3}$

Table C.2: Values of the basis function integrals of type (C.2.4) for all powers p of the linear term and all elements j for which the integral is non-zero.

and for $j=(i+1)$, we have that

$$\begin{aligned} I^{(p)}_{B_{i(i+1)}} &= \int_{x_i}^{x_{i+1}} -\frac{1}{d_{i+1}} p(x-x_{i+1})^{p-1}\frac{(x-x_i)}{d_{i+1}}dx \\ &= \frac{1}{d_{i+1}^2}\int_{x_i}^{x_{i+1}} (x-x_{i+1})^p dx \\ &= \frac{(-1)^{p+1}}{(p+1)}(d_{i+1})^{p-1}\ . \end{aligned}$$

These expressions are evaluated for powers $p=1,2$ in table C.2; note that this term does not occur for $p=0$.

Third case For the final type of integral (C.2.5), the central case $i=j$ for powers $p=0,1,2$ is

$$\begin{aligned} I^{(p)}_{C_{ii}} = &\int_{x_{i-1}}^{x_i} \frac{1}{d_i}(x-x_i)^p\frac{1}{d_i}dx \\ &+\int_{x_i}^{x_{i+1}} \Big(-\frac{1}{d_{i+1}}\Big)(x-x_i)^p\Big(-\frac{1}{d_{i+1}}\Big)dx\ . \end{aligned}$$

The first (left-hand) term is

$$\begin{aligned} \int_{x_{i-1}}^{x_i} \frac{1}{d_i}(x-x_i)^p\frac{1}{d_i}dx &= \frac{1}{d_i^2(p+1)}\Big[(x-x_i)^{p+1}\Big]_{x_{i-1}}^{x_i} \\ &= \frac{(-1)^p}{(p+1)}(d_i)^{p-1} \end{aligned}$$

and similarly, the second (right-hand) term is

$$\int_{x_i}^{x_{i+1}} \Big(-\frac{1}{d_{i+1}}\Big)(x-x_i)^p\Big(-\frac{1}{d_{i+1}}\Big)dx = \frac{1}{(p+1)}(d_{i+1})^{p-1}\ .$$

$I^{(p)}_{C_{ij}}$	$j=(i-1)$	$j=i$	$j=(i+1)$
$p=0$	$-\frac{1}{d_i}$	$\frac{1}{d_i}+\frac{1}{d_{i+1}}$	$-\frac{1}{d_{i+1}}$
$p=1$	$-\frac{1}{2}$	$-\frac{1}{2}+\frac{1}{2}$	$\frac{1}{2}$
$p=2$	$-\frac{d_i}{3}$	$\frac{d_i}{3}+\frac{d_{i+1}}{3}$	$-\frac{d_{i+1}}{3}$

Table C.3: Values of the basis function integrals of type (C.2.5) for all powers p of the linear term and all elements j for which the integral is non-zero.

The off diagonal components are as follows: for $j=(i-1)$, we have

$$\begin{aligned} I^{(p)}_{C_{i(i-1)}} &= \int_{x_{i-1}}^{x_i} \frac{1}{d_i}(x-x_{i-1})^p\Big(-\frac{1}{d_i}\Big)dx \\ &= -\frac{1}{d_i^2(p+1)}\Big[(x-x_{i-1})^{p+1}\Big]_{x_{i-1}}^{x_i} \\ &= -\frac{1}{(p+1)}(d_i)^{p-1} \end{aligned}$$

and similarly, for $j=(i+1)$, we have

$$\begin{aligned} I^{(p)}_{C_{i(i+1)}} &= \int_{x_i}^{x_{i+1}} \Big(-\frac{1}{d_{i+1}}\Big)(x-x_{i+1})^p\frac{1}{d_{i+1}}dx \\ &= \frac{(-1)^{p+1}}{(p+1)}(d_{i+1})^{p-1}\ . \end{aligned}$$

These expressions are evaluated for all relevant $p=0,1,2$ in table C.3

Appendix D

Bayes linear emulator code

This appendix presents the general code used for fitting emulators throughout the thesis (e.g. the ocean simulator example from Section 2.6, and the sequential design examples from Chapters 3, 4 and 5). Box D.1 shows the definition of the class 'BysLinEm.m'; the properties of this class store the prior specification and adjusted components of a multi-output Bayes linear emulator, and the methods allow us to adjust our prior beliefs in the light of observations on the simulator $f(.)$ and the system $y(.)$, and to make adjusted predictions for both $f(.)$ and $y(.)$.

Listing D.1: Class 'BysLinEm.m': further details of the methods listed are provided in Sections D.1 to D.4.

```
classdef BysLinEm
    % BysLinEm: Bayes Linear emulator class

    properties
        % Properties
        C           % Covariance function object
        M           % Mean function object
        F           % Attached data object
        nF          % Number of outputs
        X           % Attached input design object
        nD          % Number of data pts
        iUpd        % indicates whether the emulator has been ...
            updated
        Q           % Structure containing beliefs about Q
        %-----------------------------
        % Discrepancy
        Dsc % Discrepancy
    end

    methods
        %% Constructor
        function obj = BysLinEm(ui)
            % Constructor: assign user input
            obj.C = CovFun(ui.C);
```

```
24              obj.M = MeanFun(ui.M);
25              obj.iUpd = false;
26          end
27          %% Update
28          obj = BLUpd(obj,F,X);
29          %% Predict f
30          [EF_Fp,VarF_Fp] = BLPred(obj,pX);
31          %% Compute a
32          CovF_Fp = BLAdjCov(obj,X,Xp);
33          %% Update discrepancy
34          obj = BLDscUpd(obj,Z,X);
35          %% Predict system
36          [Ez_Y,Varz_Y] = BLPredSys(obj,X);
37          %% Predict \bar{f}
38          [EF_fb,VarF_fb] = BLAvg(obj,pX);
39      end
40
41  end
```

Some of the most important properties of this class are listed below:

- C: object of class 'CovFun', containing model-specific details of the covariance function specification;
- M: object of class 'MeanFun', containing model-specific details of the mean function specification;
- F: matrix containing the data used for the update;
- X: structure containing the inputs used for the update;
- Dsc: object containing information about the discrepancy specification.

The functions which update the model components and generate predictions are detailed in Sections D.1 to D.4.

D.1 Updating the emulator

The function 'BLUpd.m' shown in Box D.2 updates the emulator by computing and storing some quantities which will be required we generating adjusted predictions at new setting of the model inputs (predictions are generated as outlined in Section 2.2.2).

Listing D.2: Function 'BLUpd.m': updates emulator components using the simulator runs 'F' obtained at input settings 'X'.

```
function obj = BLUpd(obj,F,X)
% Update: Create static objects needed for update- exploit kronecker
% product structure as far as possible

% Dimensionality of output
nF = size(F,1);
nD = size(F,2);
% Attach data
obj.F = F;
obj.X = X;
obj.nF = nF;
obj.nD = nD;
% Basis matrix
obj.M.G = obj.M.Mean(X);
obj.M.nG = size(obj.M.G,1);
% Data mean
EF = obj.M.Eb*obj.M.G;

% Inverse data covariance
rG = kron(obj.M.G',eye(nF));
rVb = reshape(obj.M.Vb,[nF*obj.M.nG,nF*obj.M.nG]);

% Var[F] = G*Var[b]*G' + Var[U]
% This code computes Var[F]^{-1} efficiently
% Get V_{\theta{}} and V_{x}
[Cx,Ci] = obj.C.Cov(X,X);
Cx = Cx + (obj.C.Prm.ngX).*eye(size(Cx));
Ci = Ci + (0).*eye(size(Ci));
nX = size(Cx,1); nT = size(Ci,1);
% Create Kronecker matrices
iCx = pinv(Cx);
iCtf = kron(pinv(Ci),pinv(obj.C.Prm.V));
% Create Var[U]^{-1}*rG
iVUrG = nan(nF*nD,nF*obj.M.nG);
for iG = 1:(nF*obj.M.nG)
    tPrd = iCtf*reshape(rG(:,iG),[nF*nT,nX])*iCx';
    iVUrG(:,iG) = tPrd(:);
end
% Create \tilde{V}_{b}
iVbtil = (rVb\eye(size(rVb)) + iVUrG'*rG)\eye(nF*obj.M.nG);
% Create W = Var[F]^{-1}[F-E[F]]
iVUFEF = iCtf*reshape(F(:)-EF(:),[nF*nT,nX])*iCx';
W = iVUFEF(:) - iVUrG*(iVbtil*(iVUrG'*(F(:)-EF(:))));
% Construct E_{F}[b]
obj.M.EFb = reshape(obj.M.Eb(:)+ rVb*rG'*W,[obj.nF,obj.M.nG]);
% Construct Var_{F}[b]
rGtiVUrG = rG'*iVUrG;
obj.M.VFb = reshape(rVb - rVb*(rGtiVUrG - ...
    rGtiVUrG*iVbtil*rGtiVUrG')*rVb',[nF,obj.M.nG,nF,obj.M.nG]);

% store
obj.C.iVbtil = iVbtil;
obj.C.iVUrG = iVUrG;
obj.C.iCx = iCx;
obj.C.iCtf = iCtf;
obj.C.W = W;

% Indicate that update has ocurred
obj.iUpd = true;

return
```

The input 'obj' is an object of class BysLinEm, the simulator runs to be used for the update are denoted by 'F', and the object containing the corresponding input settings is denoted by 'X'.

The code computes $\mathrm{Var}\left[F\right]^{-1}$ and $\mathrm{Var}\left[F\right]^{-1}\left[F-\mathrm{E}\left[F\right]\right]$ efficiently by exploiting the Kronecker product structure imposed on the covariance matrix (see Section 2.2.3). The inverse covariance matrix $\mathrm{Var}\left[F\right]^{-1}$ is stored in terms of the inverses of the components of the Kronecker product, reducing the amount of memory that must be allocated to this object; its structure is then again exploited when generating predictions for new input settings (see Section D.2). Note that this code assumes that the covariance matrix has a three-way Kronecker product structure, where $\mathrm{Var}\left[F\right] = V_f \otimes V_a \otimes V_b$ and V_a relates to the simulator inputs V_b relates to the system inputs and V_f relates to the simulator outputs.

D.2 Computing adjusted simulator predictions

The function 'BLPred.m' shown in Box D.3 generates predictions from the emulator at new input settings, using the quantities computed and stored by the function 'BLUpd.m'.

Listing D.3: Function 'BLPred.m': computes adjusted predictive moments $\mathrm{E}_F\left[f_i\left(\theta_k\right)\right]$ and $\mathrm{Cov}_F\left[f_i\left(\theta_k\right), f_j\left(\theta_l\right)\right]$ at the input settings 'pX' using the emulator 'obj'.

```
function [EF_F,VarF_F] = BLPred(obj,pX)
% Predict: predict the function value at a new set of input
% locations

if obj.iUpd % check whether updated
    % Covariance function
    [Cx,Ci] = obj.C.Cov(pX,obj.X);
    nXp = size(Cx,1); nTp = size(Ci,1);
    nX = size(Cx,2); nT = size(Ci,2);
    nDp = nXp*nTp;
    Ctf = kron(Ci,obj.C.Prm.V);

    % Build mean prediction
    % E_{F}[u_{i}(x)]
    rEUp = Ctf*reshape(obj.C.W,[nT*obj.nF,nX])*Cx';
    EUp = reshape(rEUp,obj.nF,nDp);
    % E_{F}[\beta_{ij}]g_{j}(x)
    Gp = obj.M.Mean(pX);
    Egb = obj.M.EFb*Gp;

    % E_{F}[f_{i}(x)]
    EF_F = Egb + EUp;
    if nargout>1
        % Covariance function
        [pCx,pCi] = obj.C.Cov(pX,pX);
        pCx = pCx + (obj.C.Prm.ngX).*eye(size(pCx));
```

```
27          pCtf = kron(pCi,obj.C.Prm.V);
28          % \tilde{G} and \tilde{G}_{p}
29          rG = kron(obj.M.G',eye(obj.nF));
30          rGp = kron(Gp',eye(obj.nF));
31          % Var[b]
32          rVb = reshape(obj.M.Vb,[obj.nF*obj.M.nG,obj.nF*obj.M.nG]);
33          % Gp*Var_{F}[b]*Gp'
34          rGpVarFbGp = rGp*reshape(obj.M.VFb,...
35                                   [obj.nF*obj.M.nG,obj.nF*obj.M.nG])*rGp';
36          % Gp*Cov_{F}[b,u(x)]
37          CovUpUiVarUrG = kron(Cx*obj.C.iCx,Ctf*obj.C.iCtf)*rG;
38          rGpCovFbUp = ...
                -(rGp*(rVb*(rG'*kron(obj.C.iCx*Cx',obj.C.iCtf*Ctf'))) ...
                -...
39                   (rGp*(rVb*(rG'*(obj.C.iVUrG*obj.C.iVbtil))))*...
40                                  CovUpUiVarUrG');
41          % Var_{F}[u(x)]
42          rVarFUpUp = kron(pCx,pCtf) -...
43                       (kron(Cx*obj.C.iCx*Cx',Ctf*obj.C.iCtf*Ctf') -...
44                      CovUpUiVarUrG*obj.C.iVbtil*CovUpUiVarUrG');
45
46          % Var_{F}[f(x)]
47          VarF_F = reshape(rGpVarFbGp + rGpCovFbUp +...
48                           rGpCovFbUp' + ...
                                rVarFUpUp,[obj.nF,nDp,obj.nF,nDp]);
49      end
50  else
51      error('Update emulator before performing other operations')
52  end
53
54  return
```

Again, the input 'obj' is an object of class BysLinEm, and the code first checks whether this emulator has been updated (by checking the property 'iUpd') before it attempts to construct a prediction. If this test is passed, the code attempts to evaluate the adjusted expectations $\mathrm{E}_F\left[f_i\left(\theta_k\right)\right]$ and covariances $\mathrm{Cov}_F\left[f_i\left(\theta_k\right), f_j\left(\theta_l\right)\right]$ at a new set of inputs (passed in as the structure 'pX').

D.3 Updating using system data

Once the components of the emulator for $f(.)$ have been updated, the function 'BLDscUpd.m' (shown in Box D.4) can be used to jointly update our beliefs about the simulator and the discrepancy $\delta(.)$, as described in Section 2.4.2.

Listing D.4: Function 'BLDscUpd.m': jointly updates simulator and discrepancy components using the system data 'Z' obtained at input settings 'X'.

```
1  function obj = BLDscUpd(obj,Z,X)
2  % BLDscUpd: Update the discrepancy using system data
3
4  % # outputs
```

```
nF = obj.nF;
% # basis funs
nG = size(obj.Dsc.M.Eb,2);

% Simulator evaluated at best input
if 1
    % Evaluate at a specific simulator input setting
    [Efh,Varfh] = obj.BLPred(X);
else
    % Propagate uncertainty about best input setting
    [Efh,Varfh] = obj.BLFhPred(X);
end

% E[\delta{}(X)]
obj.Dsc.M.G = obj.Dsc.M.Mean(X);
Edel = obj.Dsc.M.Eb*obj.Dsc.M.G;
% E[Z]
EZ = EFh + Edel;
nD = size(Edel,2);

% Var[\delta{}(X)]
[Cx,Ci] = obj.Dsc.C.Cov(X,X);
Cx = Cx + obj.Dsc.C.Prm.ngX.*eye(nD);
VarU = kron(Cx,obj.Dsc.C.Prm.V);
rG = [];
for iD = 1:size(obj.Dsc.M.G,2)
    rG = [rG;kron(obj.Dsc.M.G(:,iD)',eye(nF))];
end
rVb = reshape(obj.Dsc.M.Vb,[nF*nG,nF*nG]);
% Var[Z]
Vardel = VarU + rG*rVb*rG';
VarZ = reshape(VarFh,[nF*nD,nF*nD]) + Vardel;
% store
obj.Dsc.C.iVF = VarZ\eye(size(VarZ));
obj.Dsc.C.W = obj.Dsc.C.iVF*(Z(:)-EZ(:));

% Adjust paramaeters
% E_{Z}[b]
obj.Dsc.M.EFb = reshape(obj.Dsc.M.Eb(:) + ...
    rVb*(rG'*obj.Dsc.C.W),[nF,nG]);
% Var_{Z}[b]
obj.Dsc.M.VFb = reshape(rVb - ...
    rVb*((rG'*obj.Dsc.C.iVF)*rG)*rVb,[nF,nG,nF,nG]);

% store inputs
obj.Dsc.X = X;
% store data
obj.Dsc.F = Z;

return
```

The code computes the moments $\mathrm{E}\left[\hat{f}_i\left(b_k\right)\right]$ and $\mathrm{Cov}\left[\hat{f}_i\left(b_k\right), \hat{f}_j\left(b_l\right)\right]$ (either using a user-supplied best input setting a^* or by propagating beliefs about a^* through the emulator), and $\mathrm{E}\left[\delta_i\left(b_k\right)\right]$ and $\mathrm{Cov}\left[\delta_i\left(b_k\right), \delta_j\left(b_l\right)\right]$. These components are then combined with a variance specification for the measurement error to obtain the matrix $\mathrm{Var}\left[z\right]$. The inverse covariance $\mathrm{Var}\left[z\right]^{-1}$ and the product $\mathrm{Var}\left[z\right]^{-1}\left[z - \mathrm{E}\left[z\right]\right]$ are stored along with the adjusted moments of the parameters $\beta^{(\delta)}$, to enable fast computation of adjusted predictions. As when updating the emulator (Section D.1), the Kronecker product structure of the system data covariance matrix is exploited

to reduce the memory required to store these components.

D.4 Computing adjusted system predictions

After updating using system data (Section D.3), the function 'BLPredSys.m' (Box D.5) can be used to generate adjusted predictions for the system $y(.)$ (as discussed in Section 2.4.2).

Listing D.5: Function 'BLPredSys.m': computes adjusted predictive moments $\mathrm{E}_z[y_i(b_k)]$ and $\mathrm{Cov}_z[y_i(b_k), y_j(b_l)]$ for the system.

```
function [EZY,VarZY] = BLPredSys(obj,X)
% BLPredSys: Predict the system at inputs X- including both ...
    emulator and
% discrepancy

% Compute prior moments of \hat{f}(x)
if 1
    % Evaluate at a specific simulator input setting
    [Efh,Varfh] = obj.BLPred(X);
else
    % Propagate uncertainty about best input setting
    [Efh,Varfh] = obj.BLFhPred(X);
end

%------------------------------------------------
% Adjust simulator moments by data

% Cov_F[\hat{f}(Xp),\hat{f}(X)]
nDp = size(Efh,2);
nZ = size(obj.Dsc.F,2);
rCovFfhZ = ...
    reshape(obj.BLAdjCov(X,obj.Dsc.X),[obj.nF*nDp,obj.nF*nZ]);
% E[\hat{f}(Xp)]
rEZfh = Efh(:) + rCovFfhZ*obj.Dsc.C.W(:);
% Var[\hat{f}(Xp)]
rVarZfh = reshape(Varfh,[obj.nF*nDp,obj.nF*nDp]) - ...
    rCovFfhZ*obj.Dsc.C.iVF*rCovFfhZ';

%------------------------------------------------
% Predict discrepancy at new inputs

% Basis
Gp = obj.Dsc.M.Mean(X);
rGp = [];
for iD = 1:nDp
    rGp = [rGp;kron(Gp(:,iD)',eye(obj.nF))];
end
rG = [];
for iD = 1:size(obj.Dsc.M.G,2)
    rG = [rG;kron(obj.Dsc.M.G(:,iD)',eye(obj.nF))];
end
nG = size(obj.Dsc.M.Eb,2);
rVb = reshape(obj.Dsc.M.Vb,[obj.nF*nG,obj.nF*nG]);

% Cov[U_d(Xp),U_d(X)]
[Cx,¬] = obj.Dsc.C.Cov(X,obj.Dsc.X);
rCovUpU = kron(Cx,obj.Dsc.C.Prm.V);
```

```
45 % Cov[U_d(Xp),U_d(X)]
46 [Cx,¬] = obj.Dsc.C.Cov(X,X);
47 Cx = Cx + obj.Dsc.C.Prm.ngX.*eye(nDp);
48 rCovUpUp = kron(Cx,obj.Dsc.C.Prm.V);
49 % E[\delta{}(x)]
50 rEZU = rCovUpU*obj.Dsc.C.W(:);
51 rEZGb = reshape(obj.Dsc.M.EFb*Gp,[obj.nF*nDp,1]);
52 rEZdel = rEZGb + rEZU;
53
54 % Var_Z[\delta{}(Xp)]
55 rVarZdel = (rGp*rVb*rGp' + rCovUpUp) - (rGp*rVb*rG' + ...
       rCovUpU)*(obj.Dsc.C.iVF*(rGp*rVb*rG' + rCovUpU)');
56
57 %-----------------------------------------------
58 % Collate
59
60 % Cov_Z[\delta{}(Xp),\hat{f}(Xp)]
61 CovZdelfh = -(rGp*rVb*rG' + rCovUpU)*(obj.Dsc.C.iVF*rCovFfhZ');
62
63 % sum
64 EZY = reshape(rEZdel+rEZfh,[obj.nF,nDp]);
65 VarZY = reshape(rVarZdel+rVarZfh+CovZdelfh+CovZdelfh',...
66                       [obj.nF,nDp,obj.nF,nDp]);
67
68 return
```

Again, this code uses the Kronecker product structure of the covariance matrices to reduce the memory required for the calculations.

Appendix E

Sequential design code

This appendix contains the code used to implement the example from Section 4. All code is written and run using the Matlab software package. The box E.1 shows the Matlab class 'Risk.m'; the properties of this class specify parameter settings for the example and store results as they are generated, and the methods implement the various steps of the procedure outlined in Section 3.4 for this example.

Listing E.1: Risk class: further details of the sub-functions are provided in Section D.2.

```
classdef Risk
    % Risk: properties and classes for the risk emulation procedure

    properties
        % model under analysis
        Mdl
        % design domain
        Dmn
        % loss function params
        Ls
        % emulator for risk
        Em
        % control parameters
        Cnt
    end

    methods
        %% Constructor
        function obj = Risk(ui)
            % attach model
            obj.Mdl = ui.Mdl;
            % domain
            obj.Dmn = ui.Dmn;
            % loss fun
            obj.Ls = ui.Ls;
            % control params
            obj.Cnt = ui.Cnt;
            % emulator
```

```
29              try
30                  obj.Em = ui.Em;
31              catch
32                  obj.Em = cell(1,obj.Cnt.nStg);
33              end
34          end
35          %% Convert risk input to model input
36          rXj = RskXjToDesignXj(obj,Xj,i,j);
37          %% \rho_{j}^{t}[.]
38          rjt = rhojt(obj,Xj,i,j);
39          %% Generate candidtae designs \tilde{d}_{j}
40          [rjs,Xj] = rhojs(obj,Xj,i,j);
41          %% Approximate \rho_{j}[.]
42          [E_rhoj,Var_rhoj] = rhoj(obj,Xj,i,j);
43          %% c_{j}(.)
44          c = cob(obj,Xj,i,j);
45          %% Generate design for emulator update
46          Xj = GenDesign(obj,nD,i,j);
47          %% Fit risk emulator
48          obj = FitRiskModel(obj,i,j);
49          %% Approximate Var[s_{j}^{(i)}[.]]
50          obj = AssessVarMin(obj,i,j)
51      end
52
53  end
```

The Risk class has the following properties:

- Mdl: object of class 'Model.m', which stores parameter settings and model updating routines;
- Dmn: stores domain size specifications for the design problem;
- Ls: stores parameter and cost specifications for the loss function;
- Cnt: stores control parameter settings for the risk emulation procedure;
- Em: $i \times j$ cell array of objects of class 'BysLinEm.m', which stores emulator parameter specification and update information.

In the Sections E.1 to E.4, we present a selection of functions which operate on this class.

E.1 Fit risk model

The main function which operates on objects of this class is 'FitRiskModel.m' (Box E.2); this function performs all of the steps required to fit an emulator to the risk function (as outlined in Section 3.4.4).

Listing E.2: Function 'FitRiskModel.m': fits an emulator to the risk at wave i and stage j (as outlined in Section 3.4.4).

```
function obj = FitRiskModel(obj,i,j)
% FitRiskModel: fit model to the risk at wave i, stage j

% output to command line
fprintf('Beginning emulator fit at stage %g, wave %g...\n',j,i)

%% Assess uncertainty about the minimum

if j<obj.Cnt.nStg
    % if j<n: assess variation in the minimum samples at stage (j+1)
    if 1
        obj = obj.AssessVarMin(i,j+1);
    else
        obj.Em{i,j+1}.C.Prm.Var_rjs = (0.01)^2;
    end
end

%% Generate data for risk model fit

% regression data
rXj = obj.GenDesign(obj.Cnt.Fit{i,j}.nRg,i,j);       % x_{j}
[rFj,Var_rFj] = obj.rhoj(rXj,i,j);                   % ...
    \rho_{j}[x_{j}]
% fit data
fXj = obj.GenDesign(obj.Cnt.Fit{i,j}.nFt,i,j);       % x_{j}
[fFj,Var_fFj] = obj.rhoj(fXj,i,j);                   % ...
    \rho_{j}[x_{j}]
% validation data
vXj = obj.GenDesign(obj.Cnt.Fit{i,j}.nVl,i,j);       % x_{j}
[vFj,Var_vFj] = obj.rhoj(vXj,i,j);                   % ...
    \rho_{j}[x_{j}]

%% Initialise emulator structure

% mean
ui.M.Nm = obj.Cnt.Fit{i,j}.g.Nm;
ui.M.Nmb = obj.Cnt.Fit{i,j}.g.Nmb;
% attach risk stuff
ui.M.Rsk = obj;
% delete emulators
for iI = 1:i
    for iJ = 1:obj.Cnt.nStg
        if ¬((iI==i)&&(iJ==(j+1)))
            ui.M.Rsk.Em{iI,iJ} = [];
        end
    end
end
ui.M.Prm.i = i; ui.M.Prm.j = j;
ui.M.Prm.dh = obj.Cnt.Fit{i,j}.dh;

% covariance
ui.C.Nm = obj.Cnt.Fit{i,j}.c.Nm;
ui.C.Nmb = obj.Cnt.Fit{i,j}.c.Nmb;
ui.C.Nmbb = obj.Cnt.Fit{i,j}.c.Nmbb;
% attach risk stuff
ui.C.Rsk = obj;
% delete emulators
ui.C.Rsk.Em = cell(i,obj.Cnt.nStg);
% stage and wave
ui.C.Prm.i = i; ui.C.Prm.j = j;
% nugget variance
ui.C.Prm.ngX = obj.Cnt.Fit{i,j}.ngX;

% initialise
```

```
obj.Em{i,j} = BysLinEm(ui);

%% Initialise regression components

% Var[\eps]
ve = obj.Cnt.Fit{i,j}.ve;
% g(x_{j})
G = obj.Em{i,j}.M.fG(rXj);
% #s inputs and outputs
nG = size(G,1); nF = size(rFj,1);

% E_0[b] and Var_0[b]
Eb0 = zeros(nF*nG,1);
Vb0 = eye(nF*nG);

% reshape G
rG = kron(G',eye(nF));
% Var[b]
Vb = ((rG'*((ve.*eye(size(Var_rFj)))\rG)) + Vb0\eye(nG))\eye(nG*nF);
obj.Em{i,j}.M.Vb = Vb;
% E[b]
Eb = Vb*(rG'*((ve.*eye(size(Var_rFj)))\rFj(:)) + Vb0\Eb0);
obj.Em{i,j}.M.Eb = Eb;

% \hat{\rho}_{j}[.]
rFjh = obj.Em{i,j}.M.Eb'*G;

%% Empirically fix Var[r(x)]

% \hat{r}(x_{j})
Rj = rFj-rFjh;
% Var[r(x)]
obj.Em{i,j}.C.Prm.V = cov(Rj');
% measurement error term
obj.Em{i,j}.C.Prm.erX = Var_fFj;

%% Cross-validate correlation parameters and update

obj.Em{i,j} = ...
    obj.Em{i,j}.CrossValidate(fFj,fXj,obj.Cnt.Fit{i,j}.Cvl);

%% Check fitted model against validation points

% E_{R}[\rho_{j}[.]], Var_{R}[\rho_{j}[.]]
[ER_vFj,VarR_vFj] = obj.Em{i,j}.BLPred(vXj);
vR_vFj = diag(VarR_vFj);
% (\rho_{j}[.]-E_{R}[\rho_{j}[.]])./Var_{R}[\rho_{j}[.]]^{1/2}
s = (vFj-ER_vFj)./sqrt(vR_vFj' +diag(Var_vFj)');

% output to command line
fprintf('%.3f%% of points outside 3 std. dev. error bars at\n ...
    stage %g, wave %g.\n',...
              sum(abs(s)>3).*100./numel(s),j,i);
fprintf('Emulator fit at stage %g, wave %g complete.\n',j,i)

%% if i>1: compare emulator predictions

if i>1
    % loop over waves
    ER_vFj = nan(i,obj.Cnt.Fit{i,j}.nVl);
    vR_vFj = nan(i,obj.Cnt.Fit{i,j}.nVl);
    for iI = 1:i
        [ER_vFj(iI,:),VarR_vFj] = obj.Em{iI,j}.BLPred(vXj);
        vR_vFj(iI,:) = diag(VarR_vFj);
    end
    % command line
    fprintf('Wave %g compared to wave %g:\n ',i,i-1)
    % output
```

```
128       [(ER_vFj(2,:)'-3.*sqrt(vR_vFj(2,:))'-ER_vFj(1,:)')./...
129                                 (sqrt(vR_vFj(1,:))'),...
130        (ER_vFj(2,:)'+3.*sqrt(vR_vFj(2,:))'-ER_vFj(1,:)')./...
131                                 (sqrt(vR_vFj(1,:))')]
132   end
133
134   return
```

The code initialises the structure 'obj.Em{i,j}' as an object of class BysLinEm, fixing names and parameters of basis and covariance functions using control parameters supplied from outside the function. Three sets of risk evaluations are generated at the top of the file: one for an initial regression to fix the prior moment specification for the basis parameters, one for a joint regression/residual update, and one for checking the fitted model. The regression is performed, and the prior specification for the regression components $\alpha^{(i)}$ and the marginal variance for the residual process $u^{(i)}(.)$ are fixed using the results. The cross validation is then carried out, and the correlation parameters of the covariance function are fixed. Finally, the regression and residual components are jointly updated (using the function 'BLUpd.m' presented in Section D.1), and predictions from the fitted emulator are compared with the validation data set generated earlier. For later waves ($i > 1$), the function also compares the predictions from the emulator at this wave with the predictions from the emulator at the previous wave, to check that they overlap.
Some of the sub-functions called from this function are outlined in the following sections: Section E.2 discusses 'GenDesign.m', which generates risk input sets used for fitting the emulators; Section E.3 presents 'rhoj.m' and 'rhojt.m', which generate risk evaluations at particular input settings; Section E.4 presents 'rhojs.m', which generates candidate designs for a fitted emulator.

E.2 Generate risk inputs

Box E.3 shows the function 'GenDesign.m', which is used to generate risk input sets for fitting the emulators at all stages and waves (as outlined in Section 3.4.11).

Listing E.3: Function 'GenDesign.m': generates risk input sets of size 'nD' at wave i, stage j.

```
function Xj = GenDesign(obj,nD,i,j)
% GenDesign: Generate a candidate design for the

if (i==1)||(¬obj.Cnt.DsgSample)
    %% Generate design in a space-filling way
    % setup cell
    Xj = cell(1,j);
    % Latin Hypercube
    % # d_j
    ndj = nan(j,1);
    for iJ = 1:j
        ndj(iJ) = size(obj.Dmn.L{i,iJ}.d,1);
    end
    % # w_j
    nwj = nan(j,1);
    for iJ = 1:j
        nwj(iJ) = size(obj.Dmn.L{i,iJ}.w,1);
    end
    % latin hypercube
    LHD = lhsdesign(nD,sum(ndj)+sum(nwj));

    % w_[j]
    for iJ = 1:j
        Xj{1,iJ}.w = bsxfun(@plus,obj.Dmn.L{i,iJ}.w(:,1)',...
                    bsxfun(@times,diff(obj.Dmn.L{i,iJ}.w,1,2)',...
                                LHD(:,1:nwj(iJ))));
        LHD = LHD(:,(nwj(iJ)+1):end);
    end
    % d_[j]
    for iJ = 1:j
        Xj{1,iJ}.d = bsxfun(@plus,obj.Dmn.L{i,iJ}.d(:,1)',...
                    bsxfun(@times,diff(obj.Dmn.L{i,iJ}.d,1,2)',...
                                LHD(:,1:ndj(iJ))));
        LHD = LHD(:,(ndj(iJ)+1):end);
    end

    % z_[j]
    for iJ = 1:j
        Xj{1,iJ}.z = cell(nD,1);
    end
    for iD = 1:nD
        % model inputs
        mx = obj.RskXjToModelXj(Xj,iD,i,j);
        % sample z_j
        zj = obj.Mdl.GenerateData(mx,1);
        % store
        ct = 1;
        for iJ = 1:j
            Xj{1,iJ}.z{iD} = zj(ct:(ct+mx.nz(iJ)-1));
            ct = ct+mx.nz(iJ);
        end
    end
else
    %% Generate design in interesing parts of the space
    % initialise cell
    Xj = cell(1,j);
    Xj{1,1}.d = nan(nD,1);
    % # w_j
    nwj = nan(j,1);
    for iJ = 1:j
        nwj(iJ) = size(obj.Dmn.L{i,iJ}.w,1);
    end
    % Latin hypercube
    LHD = lhsdesign(nD,sum(nwj));
    % loop over stages
    for iJ = 1:j
        % sample d_{j}^* and \rho_{j}^{*}[.]
        [rjs,Xj] = obj.rhojs(Xj,i-1,iJ);
```

```
69          % sample w_j
70          Xj{1,iJ}.w = bsxfun(@plus,obj.Dmn.L{i,iJ}.w(:,1)',...
71                          bsxfun(@times,diff(obj.Dmn.L{i,iJ}.w,1,2)',...
72                                  LHD(:,1:nwj(iJ))));
73          LHD = LHD(:,(nwj(iJ)+1):end);
74          % if iJ>1: screen out points we don't care about
75          if iJ>1
76              % \rho_{j}^{t}[.]
77              rjt = obj.rhojt(Xj,i,iJ-1);
78              % test
79              I = ((rjs-rjt')./sqrt(obj.Em{i-1,iJ}.C.Prm.Var_rjs))≤3;
80              % reduce design
81              for jJ = 1:iJ
82                  % d_j
83                  Xj{1,jJ}.d = Xj{1,jJ}.d(I,:);
84                  % w_j
85                  Xj{1,jJ}.w = Xj{1,jJ}.w(I,:);
86                  % z_j
87                  if jJ<iJ
88                      Xj{1,jJ}.z = Xj{1,jJ}.z(I);
89                  end
90              end
91              % output
92              fprintf('Wave %g: %g of %g points (%.2g%%) ...
                    eliminated at stage %g.\n',...
93                              i,sum(¬I),numel(I),iJ);
94          end
95          % sample z_j
96          Xj{1,iJ}.z = cell(nD,1);
97          for iD = 1:nD
98              % model inputs
99              mx = obj.RskXjToModelXj(Xj,iD,i,iJ);
100             % sample z_j
101             zj = obj.Mdl.GenerateData(mx,1);
102             % store
103             nzj = [0;cumsum(mx.nz)];
104             for jJ = 1:iJ
105                 Xj{1,jJ}.z{iD} = zj((nzj(jJ)+1):nzj(jJ+1));
106             end
107         end
108     end
109 end
110
111 return
```

This function generates risk inputs in each of the two cases discussed in Section 3.4.11: when $i = 1$, design inputs d_j are generated using a Latin hypercube to give good coverage of the design space; when $i > 1$, candidate designs are sampled from the emulator at the previous wave, and these designs are screened to eliminate those that we would be unlikely to select.

E.3 Evaluate risks

The function 'rhoj.m' (Box E.4) is used to generate risk characteristics for particular risk inputs (passed in as the structure 'Xj'). The risks are characterised as outlined in Section 3.4.4.

Listing E.4: Function 'rhoj.m': generate risk characteristics for risk input set 'Xj' at wave i, stage j.

```
function [E_rhoj,Var_rhoj] = rhoj(obj,Xj,i,j)
% rhoj: Evaluate \rho_{j}[.] at the inputs supplied

if j==obj.Cnt.nStg
    %% Final stage (j=n)
    % \rho_{j}[.] = \rho_{j}^{t}[.]
    % \rho_{j}^{t}[.]
    rhojt = obj.rhojt(Xj,i,j);

    % dimensions
    nD = size(rhojt,2); nF = size(rhojt,1);

    % E[\rho_{j}[.]]
    E_rhoj = rhojt;
    % Var[\rho_{j}[.]]
    Var_rhoj = zeros(nD,nD);
else
    %% Intermediate stages (j<n)
    % \rho_{j}[.] = min[\rho_{j}^{t}[.],\rho_{j+1}^{*}[.]]
    % \rho_{j}^{t}[.]
    rhojt = obj.rhojt(Xj,i,j);
    % dimensions
    nD = size(rhojt,2);

    % \rho_{j+1}^{*}[.], d_{j+1}^{*}
    [rjs,Xjs] = obj.rhojs(Xj,i,j+1);
    % get E[\rho_{j+1}[d_{j+1}^{*}]], Var[\rho_{j+1}[d_{j+1}^{*}]]
    [E_rhojs,Var_rhojs] = obj.Em{i,j+1}.BLAvg(Xjs);
    % add on c_{j}(d_{j})
    E_rhojs = E_rhojs' + obj.cob(Xjs,i,j+1);
    % cholesky
    try
        cVar_rhojs = ...
            chol(Var_rhojs+obj.Em{i,j+1}.C.Prm.Var_rjs.*eye(nD))';
    catch
        [sU,sS,sV] = ...
            svd(Var_rhojs+obj.Em{i,j+1}.C.Prm.Var_rjs.*eye(nD));
        cVar_rhojs = sU*sqrt(sS)*sV';
    end

    % sample rhoj
    rhojs = bsxfun(@plus,E_rhojs,cVar_rhojs*randn(nD,obj.Cnt.nSmp));
    rhoj = bsxfun(@min,rhojt',rhojs);

    % E[\rho_{j}[.]]
    E_rhoj = mean(rhoj,2);
    % Var[\rho_{j}[.]]
    Var_rhoj = (rhoj*rhoj')./obj.Cnt.nSmp - E_rhoj*E_rhoj';

    % transpose
    E_rhoj = E_rhoj';
end

%% For later waves- check data against fitted risk models
if i>1
    % Emulator predictions E_{R}[\rho_{j}[.]], Var_{R}[\rho_{j}[.]]
    [ER_rjt,VarR_rjt] = obj.Em{i-1,j}.BLPred(Xj);
    VarR_rjt = diag(VarR_rjt)';
    % s(x_j)
    s = (E_rhoj-ER_rjt)./sqrt(VarR_rjt+diag(Var_rhoj)');
    % output
    fprintf('%.3f%% of new data outside 3 std dev error bars at ...
        wave %g.\n',...
            (sum(abs(s(:))>3)./numel(E_rhoj)).*100,i-1)
```

```
62 end
63
64 return
```

This function computes risk moments in two separate cases: the final stage $j = n$, where $\rho_n = \rho_n^t$, and any other stage $j < n$, where $\rho_j = \min[\rho_j^t, \rho_{j+1}^*]$. At the final stage, we simply evaluate the terminal risk ρ_n^t at the inputs supplied in the structure 'Xj' by calling the function 'rhojt.m' (Box E.5). At any other stage, we use this function to evaluate the terminal risk at these inputs, use the function 'rhojs.m' (Box E.6) to generate candidate designs $\tilde{d}_{j+1}$ for each input setting, evaluate our approximation $\bar{r}_{j+1}^{(i)}\left[\tilde{d}_{j+1}\right] + c_{j+1}(j+1)$ at these settings, and compare these risks for each design input by sampling.

Listing E.5: Function 'rhojt.m': evaluate terminal risk for risk input set 'Xj' at stage j.

```
1  function rjt = rhojt(obj,Xj,i,j)
2  % rhojt: generate terminal risk for the inputs
3
4  % # inputs
5  nD = size(Xj{1,1}.d,1);
6
7  % initialise
8  rjt = nan(1,nD);
9  % start timing
10 tic
11 % loop over points
12 for iD = 1:nD
13     % convert to model inputs
14     mXj = obj.RskXjToModelXj(Xj,iD,i,j);
15
16     % get E_{z_[j]}[q], Var_{z_[j]}[q]
17     [Ezj_q,Varzj_q] = obj.Mdl.BayesLinearAdjust(mXj);
18     vzj_q = diag(Varzj_q);
19     % get E_{z_[j]}[\sum_{k}q_k], Var_{z_[j]}[\sum_{k}q_k]
20     Ezj_qt = ones(1,numel(Ezj_q))*Ezj_q;
21     Varzj_qt = ones(1,numel(Ezj_q))*diag(diag(Varzj_q))*...
22         ones(numel(Ezj_q),1);
23     % moments
24     mom(:,1) = Ezj_qt;
25     mom(:,2) = Ezj_qt.^2 + Varzj_qt;
26     mom(:,3) = Ezj_qt.^3 + 3.*Ezj_qt.*Varzj_qt;
27     mom(:,4) = Ezj_qt.^4 + 6.*(Ezj_qt.^2).*Varzj_qt +...
28                                         3.*(Varzj_qt.^2);
29
30     % loss components
31     % E[w(q)]
32     E_w = obj.Ls.alp(1) + obj.Ls.alp(2).*mom(:,1) +...
33                                         obj.Ls.alp(3).*mom(:,2);
34     % E[w(q).q]
35     E_wq = obj.Ls.alp(1).*mom(:,1) + obj.Ls.alp(2).*mom(:,2) +...
36                                         obj.Ls.alp(3).*mom(:,3);
37     % E[w(q).(q^2)]
38     E_wq2 = obj.Ls.alp(1).*mom(:,2) + obj.Ls.alp(2).*mom(:,3) +...
39                                         obj.Ls.alp(3).*mom(:,4);
```

```
40
41      % E[L(q,a)]
42      rjt(:,iD) = obj.Ls.C + E_wq2 - (1./E_w).*(E_wq.^2);
43
44      % count
45      if rem(iD,10)==0
46          fprintf('#')
47          if rem(iD,100)==0
48              fprintf('\n')
49              fprintf('%.1f%% of data generated in %.1f ...
                    seconds.\n',...
50                                                  100*iD/nD,toc)
51          end
52      end
53  end
54
55  return
```

E.4 Generate candidate designs

The function 'rhojs.m' (Box E.6) is used to generate candidate designs $\tilde{d}_j$ at particular risk input settings (supplied in the structure 'Xj').

Listing E.6: Function 'rhojs.m': generate candidate designs $\tilde{d}_j$ (at wave i, stage j) at the risk input settings supplied in the structure 'Xj'.

```
1   function [rjs,Xj] = rhojs(obj,Xj,i,j)
2   % rhojs: evaluate \rho_{j}^{*}[.] at the inputs supplied (for ...
        stage j, wave
3   % i)
4
5   % # inputs
6   nD = size(Xj{1,1}.d,1);
7
8   % initialise
9   rjs = nan(nD,1);
10  ds = nan(nD,size(obj.Dmn.L{i,j}.d,1));
11  % loop over data points
12  pause(0.5)
13  parfor_progress(nD);
14  pause(0.5)
15  parfor iD = 1:nD
16      % Xj- test design
17      tXj = testXj(obj,Xj,iD,i,j);
18      % E_{R}[\rho_{j}[.]], Var_{R}[\rho_{j}[.]]
19      [ER_rj,VarR_rj] = obj.Em{i,j}.BLAvg(tXj);
20      VarR_rj = squeeze(VarR_rj);
21      % add on c_{j}(.) (obs cost)
22      ER_rj = ER_rj' + obj.cob(tXj,i,j);
23
24      % sample \rho_{j}[.]
25      try
26          cVarR_rj = chol(VarR_rj)';
27      catch
28          [sU,sS,sV] = svd(VarR_rj);
29          cVarR_rj = sU*sqrt(sS)*sV';
30      end
31      rj = ER_rj + cVarR_rj*randn(obj.Cnt.Min{i,j}.nTs,1);
```

```
32
33      % find min
34      [mrj,mi] = min(rj);
35      % attach
36      rjs(iD) = mrj;
37      ds(iD,:) = tXj{1,j}.d(mi,:);
38
39      % count
40      parfor_progress
41  end
42  pause(0.5)
43  parfor_progress(0);
44  pause(0.5)
45
46  % attach d_{j}^{*}
47  Xj{1,j}.d = ds;
48
49  return
50
51  function tXj = testXj(obj,Xj,iD,i,j)
52  % Generate space-filling design to interrogate emulator
53
54  % # d_j
55  ndj = size(obj.Dmn.L{i,j}.d,1);
56
57  % # test points
58  nTs = obj.Cnt.Min{i,j}.nTs;
59
60  % Latin hypercube
61  LHD = lhsdesign(nTs,ndj);
62
63  % replicate iD^th inputs up to stage (j-1)
64  tXj = cell(1,j);
65  for iJ = 1:(j-1)
66      % d_j
67      tXj{1,iJ}.d = repmat(Xj{1,iJ}.d(iD,:),[nTs,1]);
68      % w_j
69      tXj{1,iJ}.w = repmat(Xj{1,iJ}.w(iD,:),[nTs,1]);
70      % z_j
71      tXj{1,iJ}.z = repmat(Xj{1,iJ}.z(iD),[nTs,1]);
72  end
73
74  % attach Latin hypercube at stage j
75  tXj{1,j}.d = bsxfun(@plus,obj.Dmn.L{i,j}.d(:,1)',...
76                      bsxfun(@times,diff(obj.Dmn.L{i,j}.d,1,2)',LHD));
77
78  return
```

For each of the supplied input settings, the code generates a Latin hypercube inside the candidate design space at the previous wave, evaluates the risk emulator at each point in the Latin hypercube, and then retains as a candidate design the point which minimises the risk over this Latin hypercube. The candidate designs are re-attached to the input structure 'Xj', before being passed out of the function. This function makes use of Matlab's 'parfor' loop, which distributes the iterations of the 'for' loop over the available CPUs, cutting the overall computation time by performing calculations in parallel.

Appendix F

Bell-tower model code

In this appendix, we present a selection of the Matlab code used to implement the coupled bell-tower model example from Section 6.2. Box F.1 presents the class definition 'Model.m'; this class stores model properties and defines methods which perform operations on these properties.

Listing F.1: Class 'Model.m': stores information and performs operations for the example presented in Section 6.2.

```
classdef Model
    % Model: store parameters and perform operations for the ...
       bell-tower
    % model solver

    properties (SetAccess = private)
        Slv     % Solver (class: Solver)
        Ndc     % Numerical discrepancy (class: NumDiscrepancy)
        P       % Paramteter names and characteristics
    end

    properties (SetAccess = public)
        J       % Junction tree (class: JunctionTree)
        Grd     % Temporal grid properties
        Cnt     % Sampler control parameters
        Dmn     % Domain specification
        Eps     % Measurement error properties
    end

    methods
        %% Constructor
        function obj = Model(ui)
            obj.Slv = Solver(ui.Slv);
            obj.Ndc = NumDiscrepancy(ui.Ndc);
            obj.Eps = ui.Eps;
        end
        %% Generate graphical prior
        obj = Prior(obj,E0,Cov0);
        %% Plot solution
        PlotSolution(obj,Adj,Elm);
```

```
30      end
31
32  end
```

The properties of this class include the following:

- Slv: object of class 'Solver' (see Section F.1), which contains parameter settings for the ODE solver from Section 6.2.1, and defines methods which implement the numerical scheme outlined in Section 6.2.2;
- Ndc: object of class 'NumDiscrepancy', which stores properties of the numerical discrepancy model outlined in Section 6.1.6;
- J: object of class 'JunctionTree' (see Section F.2), which stores information about the structure of the junction tree and defines methods for adjusting the whole graph given observations of certain nodes.

F.1 Solver class

The class 'Solver.m' (Box F.2) stores parameter settings for the ODE system outlined in Section 6.2.1 and defines methods which implement the numerical scheme developed in Section 6.2.2.

Listing F.2: Class 'Solver.m': stores parameters for the bell-tower ODE system (Section 6.2.1), and implements numerical schemes (Section 6.2.2).

```
1  classdef Solver
2      % Solver for the bell-tower system of ODEs
3
4      properties (SetAccess = private)
5          Prm     % Fixed solver params
6          Nms     % Input names
7      end
8
9      methods
10         %% Constructor
11         function obj = Solver(ui)
12             obj.Prm = ui.Prm;
13             obj.Nms = ui.Nms;
14         end
15         %% Derivative function
16         % du_i/dt(t) = f_i(t,u(t),\theta)
17         function [du,A,b] = f(obj,phi)
18             % # data inputs
19             nD = size(phi.th,2);
20             % storage
21             du.ddx = nan(2,nD);
```

```
22              du.ddth = nan(obj.Prm.B.n,nD);
23              for iD = 1:nD
24                  %----------------------
25                  % d^{2}\phi_i/dt^{2}
26                  % A
27                  cA = cell(2,2);
28                  % d^{2}x/dt^{2} equations
29                  cA{1,1} = eye(2);
30                  cA{1,2} = bsxfun(@times,obj.Prm.B.drc',...
31                      ((obj.Prm.B.m.*obj.Prm.B.rc./obj.Prm.C.M).*...
32                                  cos(phi.th(:,iD)))');
33                  % d^{2}\theta{}/dt^{2}
34                  cA{2,1} = ...
                        bsxfun(@times,cos(phi.th(:,iD)),obj.Prm.B.drc);
35                  cA{2,2} = diag(obj.Prm.B.l);
36                  % b
37                  cb = cell(2,1);
38                  % d^{2}x/dt^{2} equations
39                  cb{1} = ...
                        obj.Prm.B.drc'*((obj.Prm.B.m.*obj.Prm.B.rc./...
40                                                      obj.Prm.C.M).*...
41                      (phi.dth(:,iD).^2).*cos(phi.th(:,iD))) -...
42                      2.*phi.lam(:,iD).*phi.dx(:,iD) -...
43                      (phi.wg(:,iD).^2).*phi.x(:,iD);
44                  % d^{2}\theta{}/dt^{2} equations
45                  cb{2} = -obj.Prm.g.*sin(phi.th(:,iD));
46                  %----------------------
47                  % solve
48                  A = cell2mat(cA); b = cell2mat(cb);
49                  v = A\b;
50                  % d^{2}x/dt^{2}
51                  du.ddx(:,iD) = v(1:2);
52                  % d^{2}\theta{}/dt^{2}
53                  du.ddth(:,iD) = v(3:end);
54              end
55              % degree conversion
56              du.ddth = du.ddth;
57          end
58          %% Higher-order derivative function
59          function du = df(obj,phi)
60              % # data inputs
61              nD = size(phi.th,2);
62              % storage
63              du.ddx = nan(2,nD);
64              du.ddth = nan(obj.Prm.B.n,nD);
65              du.dddx = nan(2,nD);
66              du.dddth = nan(obj.Prm.B.n,nD);
67              du.ddddx = nan(2,nD);
68              du.ddddth = nan(obj.Prm.B.n,nD);
69              % loop over data
70              for iD = 1:nD
71                  %----------------------
72                  % f(u,\theta)
73                  % A
74                  cA = cell(2,2);
75                  % x eq.
76                  cA{1,1} = eye(2);
77                  cA{1,2} = bsxfun(@times,obj.Prm.B.drc',...
78                      ((obj.Prm.B.m.*obj.Prm.B.rc./obj.Prm.C.M).*...
79                                  cos(phi.th(:,iD)))');
80                  % \theta eq.
81                  cA{2,1} = ...
                        bsxfun(@times,cos(phi.th(:,iD)),obj.Prm.B.drc);
82                  cA{2,2} = diag(obj.Prm.B.l);
83                  % b
84                  cb = cell(2,1);
85                  % x eq.
86                  cb{1} = ...
```

```
                    obj.Prm.B.drc'*((obj.Prm.B.m.*obj.Prm.B.rc./...
87                                  obj.Prm.C.M).*...
88                   (phi.dth(:,iD).^2).*cos(phi.th(:,iD))) -...
89                   2.*phi.lam(:,iD).*phi.dx(:,iD) -...
90                   (phi.wg(:,iD).^2).*phi.x(:,iD);
91                % \theta eq.
92                cb{2} = -obj.Prm.g.*sin(phi.th(:,iD));
93                % f
94                A = sparse(cell2mat(cA)); b = cell2mat(cb);
95                iA = A\eye(size(A));
96                fv = iA*b;
97                du.ddx(:,iD) = fv(1:2);
98                du.ddth(:,iD) = fv(3:end);
99                %----------------------
100               % df/dt(u,\theta)
101               % dA/dt
102               cdA = cell(2,2);
103               % x eq.
104               cdA{1,1} = zeros(2,2);
105               cdA{1,2} = -bsxfun(@times,obj.Prm.B.drc',...
106                   ((obj.Prm.B.m.*obj.Prm.B.rc./obj.Prm.C.M).*...
107                   phi.dth(:,iD).*sin(phi.th(:,iD)))');
108               % \theta eq.
109               cdA{2,1} = ...
                      -bsxfun(@times,obj.Prm.B.drc,phi.dth(:,iD).*...
110                                  sin(phi.th(:,iD)));
111               cdA{2,2} = zeros(obj.Prm.B.n);
112               % db/dt
113               cdb = cell(2,1);
114               % x eq.
115               cdb{1} = obj.Prm.B.drc'*...
116                   ((obj.Prm.B.m.*obj.Prm.B.rc./obj.Prm.C.M).*...
117                   (2.*phi.dth(:,iD).*du.ddth(:,iD).*...
118                                 sin(phi.th(:,iD)) +...
119                   (phi.dth(:,iD).^3).*cos(phi.th(:,iD)))) -...
120                   2.*phi.lam(:,iD).*du.ddx(:,iD) -...
121                                (phi.wg(:,iD).^2).*phi.dx(:,iD);
122               % \theta eq.
123               cdb{2} = ...
                      -obj.Prm.g.*phi.dth(:,iD).*cos(phi.th(:,iD));
124               % df/dt
125               dA = cell2mat(cdA); db = cell2mat(cdb);
126               dfv = iA*(db - dA*fv);
127               du.dddx(:,iD) = dfv(1:2);
128               du.dddth(:,iD) = dfv(3:end);
129               %----------------------
130               % d^{2}f/dt^{2}(u,\theta)
131               % d^{2}A/dt^{2}
132               cddA = cell(2,2);
133               % x eq.
134               cddA{1,1} = zeros(2,2);
135               cddA{1,2} = -bsxfun(@times,obj.Prm.B.drc',...
136                             (obj.Prm.B.m.*obj.Prm.B.rc./obj.Prm.C.M)'.*...
137                                (du.ddth(:,iD).*sin(phi.th(:,iD)) ...
                                      +...
138                                (phi.dth(:,iD).^2).*cos(phi.th(:,iD))));
139               % \theta eq.
140               cddA{2,1} = -bsxfun(@times,obj.Prm.B.drc,...
141                                 du.ddth(:,iD).*sin(phi.th(:,iD)) ...
                                       +...
142                                 (phi.dth(:,iD).^2).*cos(phi.th(:,iD)));
143               cddA{2,2} = zeros(obj.Prm.B.n);
144               % d^{2}b/dt^{2}
145               cddb = cell(2,1);
146               % x eq.
147               cddb{1} = ...
                      obj.Prm.B.drc'*((obj.Prm.B.m.*obj.Prm.B.rc./...
148                                          obj.Prm.C.M).*...
```

```
149                                 2.*(du.dddth(:,iD).*phi.dth(:,iD) ...
                                        +...
150                                 du.ddth(:,iD).^2).*sin(phi.th(:,iD)) ...
                                        +...
151                                 5.*du.ddth(:,iD).*(phi.dth(:,iD).^2).*...
152                                                 cos(phi.th(:,iD)) ...
                                                        -...
153                                 (phi.dth(:,iD).^4).*sin(phi.th(:,iD))) ...
                                        -...
154                                 2.*phi.lam(:,iD).*du.dddx(:,iD) -...
155                                 (phi.wg(:,iD).^2).*du.ddx(:,iD);
156                 % \theta eq.
157                 cddb{2} = ...
                        -obj.Prm.g.*(du.ddth(:,iD).*cos(phi.th(:,iD)) ...
                        -...
158                                 (phi.dth(:,iD).^2).*sin(phi.th(:,iD)));
159                 % d^{2}f/dt^{2}
160                 ddA = cell2mat(cddA); ddb = cell2mat(cddb);
161                 ddfv = iA*(ddb - ddA*fv - 2.*dA*dfv);
162                 du.ddddx(:,iD) = ddfv(1:2);
163                 du.ddddth(:,iD) = ddfv(3:end);
164             end
165         end
166         %% Euler evolution function
167         % \hat{u}_i(t) = F_{i}(t,u(t),\theta)
168         function uh = Fel(obj,phi)
169             %----------------------
170             % Evaluate f(.)
171             f = obj.f(phi);
172             uh = phi;
173             %----------------------
174             % \hat{dx/dt}_i(t)
175             uh.dx = phi.dx + bsxfun(@times,f.ddx,phi.dt);
176             %----------------------
177             % \hat{d\theta/dt}_j(t)
178             uh.dth = phi.dth + bsxfun(@times,f.ddth,phi.dt);
179             %----------------------
180             % \hat{x}_i(t)
181             uh.x = phi.x + bsxfun(@times,uh.dx,phi.dt);
182             %----------------------
183             % \hat{\theta}_j(t)
184             uh.th = phi.th + bsxfun(@times,uh.dth,phi.dt);
185             %----------------------
186             % impose hard stop at top of swing
187             % >180
188             uh.dth(uh.th>pi) = -uh.dth(uh.th>pi);
189             uh.th(uh.th>pi) = pi;
190             % <-180
191             uh.dth(uh.th<-pi) = -uh.dth(uh.th<-pi);
192             uh.th(uh.th<-pi) = -pi;
193         end
194         %% Second-order Runge-Kutta evolution function
195         function uh = Frk2(obj,phi)
196             %----------------------
197             % Evaluate components
198             % constants
199             a21 = 1/2;
200             c2 = 1/2; b2 = 1;
201             h = phi.dt;
202             % \xi_1
203             xi1 = phi;
204             % \xi_2
205             fx1 = obj.f(xi1);
206             xi2.dx = phi.dx + a21.*bsxfun(@times,fx1.ddx,h);
207             xi2.x = phi.x + a21.*bsxfun(@times,xi1.dx,h);
208             xi2.dth = phi.dth + a21.*bsxfun(@times,fx1.ddth,h);
209             xi2.th = phi.th + a21.*bsxfun(@times,xi1.dth,h);
210             xi2.lam = phi.lam;
211             xi2.wg = phi.wg;
```

```
212             %----------------------
213             % Approximation
214             % initialise
215             uh = phi;
216             % evolve \xi_2
217             fx2 = obj.f(xi2);
218             % update
219             uh.dx = uh.dx + b2.*bsxfun(@times,fx2.ddx,h);
220             uh.x = uh.x + b2.*bsxfun(@times,xi2.dx,h);
221             uh.dth = uh.dth + b2.*bsxfun(@times,fx2.ddth,h);
222             uh.th = uh.th + b2.*bsxfun(@times,xi2.dth,h);
223         end
224         %% Second-order evolution function
225         function uh = Fel2(obj,phi)
226             %----------------------
227             % Evaluate f(.) and derivatives
228             df = obj.df(phi);
229             uh = phi;
230             %----------------------
231             % \hat{dx/dt}_i(t)
232             uh.dx = phi.dx + bsxfun(@times,df.ddx,phi.dt) +...
233                              bsxfun(@times,df.dddx,phi.dt.^2).*(1/2);
234             %----------------------
235             % \hat{d\theta/dt}_j(t)
236             uh.dth = phi.dth + bsxfun(@times,df.ddth,phi.dt) +...
237                                bsxfun(@times,df.dddth,phi.dt.^2).*(1/2);
238             %----------------------
239             % \hat{x}_i(t)
240             uh.x = phi.x + bsxfun(@times,uh.dx,phi.dt) +...
241                            bsxfun(@times,df.ddx,phi.dt.^2).*(1/2);
242             %----------------------
243             % \hat{\theta}_j(t)
244             uh.th = phi.th + bsxfun(@times,uh.dth,phi.dt) +...
245                              bsxfun(@times,df.ddth,phi.dt.^2).*(1/2);
246             %----------------------
247             % impose hard stop at top of swing
248             if any(uh.th(:)>pi)||any(uh.th(:)<-pi)
249                 pi
250             end
251             % >180
252             uh.dth(uh.th>pi) = -uh.dth(uh.th>pi);
253             uh.th(uh.th>pi) = pi;
254             % <-180
255             uh.dth(uh.th<-pi) = -uh.dth(uh.th<-pi);
256             uh.th(uh.th<-pi) = -pi;
257         end
258     end
259
260 end
```

This class definition contains a number of function definitions. The function 'f' evaluates the first-order derivatives $\frac{du}{dt}(t)$ as outlined in Section 6.2.2, and the function 'df' evaluates the corresponding higher-order derivatives (up to third-order). The input structure 'phi' contains the components of the current solution state vector and parameter specifications. The function 'Fel' uses these derivative functions to implement a first-order Euler solver, and 'Fel2' implements a second-order Euler scheme.

F.2 Junction tree class

The class 'JunctionTree.m' (Box 2.4.2) allows for specification of a junction tree graphical model (see Section 2.1.3) and implements methods which adjust the prior moments of all components by propagating information around the junction tree.

Listing F.3: Class 'JunctionTree.m': stores clique specifications and prior and adjusted moments for model components, and provides methods for adjusting all components by propagating information around the junction tree.

```
classdef JunctionTree
    % JunctionTree: class to store properties and methods for a ... Bayes
    % linear junction tree

    properties (SetAccess = private)
        % Associated graph
        G
        % Cliques
        Clq
        % # cliques
        nC
    end

    methods
        %% Constructor
        function obj = JunctionTree(ui)
            % attach graph
            obj.G = ui.G;
            % # cliques
            obj.nC = numel(ui.Clq);
            % initialise cliques
            ui.Clq(1).G = ui.G;
            obj.Clq = Clique(ui.Clq(1));
            for iC = 2:obj.nC
                ui.Clq(iC).G = ui.G;
                tic
                obj.Clq(iC) = Clique(ui.Clq(iC));
                t = toc;
                fprintf('Clique %g of %g created in %.1f ... seconds.\n',iC,obj.nC,t)
            end
        end
        %% Establish clique membership
        function mid = CliqueMembership(obj,Ndi)
            % convert input type
            if ischar(Ndi)
                iI = find(strcmp({obj.Nde.Lbl},Ndi));
            elseif isnumeric(Ndi)
                iI = Ndi;
            end
            % search through cliques
            mid = false(obj.nC,1);
            for iC = 1:obj.nC
                if ismember(iI,obj.Clq(iC).nid)
                    mid(iC) = true;
                end
            end
            mid = find(mid);
        end
```

```
49          %% Establish clique intersection
50          function cis = CliqueIntersection(obj,Cqi,Cqj)
51              % find intersection
52              cis = intersect(obj.Clq(Cqi).nid,obj.Clq(Cqj).nid);
53          end
54          %% Establish clique neighbours
55          function [cng,cis] = CliqueNeighbours(obj,Cqi)
56              % loop over cliques
57              cis = cell(obj.nC,1);
58              cng = false(obj.nC,1);
59              for iC = 1:obj.nC
60                  cis{iC} = obj.CliqueIntersection(Cqi,iC);
61                  if ¬isempty(cis{iC})
62                      cng(iC) = true;
63                  end
64              end
65              % reduce
66              cng = find(cng);
67              cis = cis(cng);
68              % delete cell itself
69              cis(cng==Cqi) = [];
70              cng(cng==Cqi) = [];
71          end
72          %% Cov[v_i,v_j]
73          Cov_vi_vj = PairwiseCov(obj,Ndi,Ndj);
74          %% Sequentially adjust
75          obj = SequentialAdjust(obj,F);
76          obj = newSequentialAdjust(obj,F);
77      end
78
79  end
```

This class has the following notable properties:

- G: contains the original DAG specification for the problem (ordering, neighbourhood structure and prior moment specification);
- Clq: $n_C \times 1$ structure array of objects of class 'Clique.m', with each corresponding to an individual clique. Each structure specifies which nodes on the original graph are members of the clique, the full covariance structure for all clique components, and which cliques are neighbours of this one.

The class 'Clique.m' is presented in Section F.2.1, and in Section F.2.2, we present the function 'SequentialAdjust.m', which performs adjustments on the junction tree as outlined in Section 2.1.3.

F.2.1 Clique class

The class 'Clique.m' stores information about an individual clique of the junction tree.

Listing F.4: Class 'Clique.m': stores information about an individual clique of the junction tree.

```
classdef Clique
    % Clique: clique class for the junction tree

    properties (SetAccess = private)
        % Clique number
        n
        % Prior Moments
        E           % Expectation of each node
        Cov         % Covariance between pairs of nodes
    end

    properties (SetAccess = public)
        % Adjusted moments
        EF        % Expectation of each node
        CovF      % Covariance between pairs of nodes
        % Fields
        Fld
        % Field indices
        nid
        % # fields
        nF
        % Neighbouring cliques
        Ngb
    end

    methods
        %% Constructor
        function obj = Clique(ui)
            % field names
            obj.Fld = ui.Fld;
            % clique number
            obj.n = ui.n;
            % # fields
            obj.nF = numel(obj.Fld);
            % corres. indices
            obj.nid = nan(obj.nF,1);
            for iF = 1:obj.nF
                obj.nid(iF) = ...
                    find(strcmp({ui.G.Nde.Lbl},obj.Fld{iF}));
            end
            % neighbours
            obj.Ngb = ui.Ngb;
            % prior expectations and covariances
            obj.E = ui.E;
            obj.Cov = ui.Cov;
        end
    end

end
```

The fields 'E' and 'Cov' are cell arrays containing the prior expectation and covariance specifications for the nodes in this clique, and the fields 'EF' and 'CovF' contain the corresponding adjusted moments once they have been computed using the function 'SequentialAdjust.m' (Section F.2.2). The field 'Ngb' specifies which cliques are neighbours of this one on the junction tree.

F.2.2 Sequential adjustment

For a given junction tree object, the function 'SequentialAdjust.m' (Box F.5) adjusts all nodes on the graph using each data point in turn, by using the structure of the junction tree to propagate information between the cliques.

Listing F.5: Function 'SequentialAdjust.m': adjust all model components by propagating information around the cliques of the junction tree.

```
function obj = SequentialAdjust(obj,F)
% SequentialAdjust: work outwards from the observed clique, and ...
    adjust
% clique components sequentially

% # data points
nF = numel(F.I);

%% Initialise adjusted moments

if isempty(obj.Clq(1).EF)
    % Loop over cliques
    for iC = 1:obj.nC
        obj.Clq(iC).EF = obj.Clq(iC).E;
        obj.Clq(iC).CovF = obj.Clq(iC).Cov;
    end
end

%% Sequential update

for iF = 1:nF
    %% Create copy for partial adjustment
    pob = obj;

    %% Data moments
    % current observed node
    fNd = F.I(iF);
    % observed clique
    allfClq = obj.CliqueMembership(fNd);
    fClq = allfClq(1);

    % D_{ik}
    Dk = cell2mat(F.Nde(fNd).D);
    % E[D_{ik}]
    dI = obj.Clq(fClq).nid==fNd;
    E_Dk = cell2mat(obj.Clq(fClq).EF{dI});
    % Var[D_{ik}]
    Var_Dk = cell2mat(obj.Clq(fClq).CovF{dI,dI});
    % Var[D_k]^{-1}
    iVar_Dk = Var_Dk\eye(size(Var_Dk));

    %% Adjust observed clique
    % Loop over elements
    for iN = 1:obj.Clq(fClq).nF
        % Cov[D_{ik},v_{jl}]
        Cov_Dk_vj = cell2mat(obj.Clq(fClq).CovF{dI,iN});
        % E_{D_k}[v_{jl}]
        EDk_vj = cell2mat(obj.Clq(fClq).EF{iN}) +...
                            Cov_Dk_vj'*(iVar_Dk*(Dk-E_Dk));
        % restore
        pob.Clq(fClq).EF{iN} =..
                mat2cell(EDk_vj,obj.G.Nde(obj.Clq(fClq).nid(iN)).nV,1);
```

```
52          for jN = 1:iN
53              % Cov[D_{ik},v_{pq}]
54              Cov_Dk_vp = cell2mat(obj.Clq(fClq).CovF{dI,jN});
55              % Cov_{D_k}[v_{jl},v_{pq}]
56              CovDk_vj_vp = cell2mat(obj.Clq(fClq).CovF{iN,jN}) -...
57                            Cov_Dk_vj'*(iVar_Dk*Cov_Dk_vp);
58              % restore
59              pob.Clq(fClq).CovF{iN,jN} = mat2cell(CovDk_vj_vp,...
60                                  obj.G.Nde(obj.Clq(fClq).nid(iN)).nV,...
61                                  obj.G.Nde(obj.Clq(fClq).nid(jN)).nV);
62              pob.Clq(fClq).CovF{jN,iN} = mat2cell(CovDk_vj_vp',...
63                                  obj.G.Nde(obj.Clq(fClq).nid(jN)).nV,...
64                                  obj.G.Nde(obj.Clq(fClq).nid(iN)).nV);
65          end
66      end
67
68      %% Propagate adjustment outwards from here
69      % initialise stopping indicator
70      Stop = false;
71
72      % initialise current clique C_i
73      cClq = fClq;
74      % initialise current nodes N_i
75      cEl = {fNd};
76      % initialise history
77      hstClq = cClq;
78      % Cov[D_k,.]
79      CFn = {eye(sum(obj.G.Nde(fNd).nV))};
80
81      % propagate
82      while ¬Stop
83          % time
84          tic
85          % Find Ngb(C_i)
86          cNgb = cell(numel(cClq),1);
87          for iC = 1:numel(cClq)
88              cNgb{iC} = obj.Clq(cClq(iC)).Ngb;
89              % eliminate ones already visited
90              cNgb{iC}(ismember(cNgb{iC},hstClq)) = [];
91          end
92
93          % Find C_i\cap{}C_j and update state
94          newCFn = [];
95          newcEl = [];
96          for iC = 1:numel(cClq)
97              for jC = 1:numel(cNgb{iC})
98                  % C_i\cap{}C_j
99                  Ci_n_Cj = ...
                        obj.CliqueIntersection(cClq(iC),cNgb{iC}(jC));
100                 % check for bad specification
101                 if isempty(Ci_n_Cj)
102                     error('No intersection between neighbouring ...
                            cliques!')
103                 end
104                 % Cov[v_{ik},{C_i\cap{}C_j}_{jl}]
105                 Cov_vi_CinCj = cell(numel(cEl{iC}),numel(Ci_n_Cj));
106                 for kC = 1:numel(cEl{iC})
107                     for lC = 1:numel(Ci_n_Cj)
108                         lI = obj.Clq(cClq(iC)).nid==cEl{iC}(kC);
109                         rI = obj.Clq(cClq(iC)).nid==Ci_n_Cj(lC);
110                         Cov_vi_CinCj{kC,lC} =...
111                                 cell2mat(obj.Clq(cClq(iC)).CovF{lI,rI});
112                     end
113                 end
114                 % Var[C_i\cap{}C_j]
115                 Var_CinCj = cell(numel(Ci_n_Cj));
116                 for kC = 1:numel(Ci_n_Cj)
117                     for lC = 1:numel(Ci_n_Cj)
```

```
                        lI = obj.Clq(cClq(iC)).nid==Ci_n_Cj(kC);
                        rI = obj.Clq(cClq(iC)).nid==Ci_n_Cj(lC);
                        Var_CinCj{kC,lC} =...
                            cell2mat(obj.Clq(cClq(iC)).CovF{lI,rI});
                    end
                end
                % update Cov[D_k,.]
                newCFn = ...
                    [newCFn;{CFn{iC}*(cell2mat(Cov_vi_CinCj)/...
                                        cell2mat(Var_CinCj))}];
                % update current elements
                newcEl = [newcEl;{Ci_n_Cj}];
            end
        end

        % Update state
        % list of visited cliques
        for iC = 1:numel(cNgb)
            hstClq = [hstClq;cNgb{iC}];
        end
        % current state
        cClq = [];
        for iC = 1:numel(cNgb)
            cClq = [cClq;cNgb{iC}];
        end
        % Cov[D_k,.]
        CFn = newCFn;
        % Elements
        cEl = newcEl;

        % Stop?
        if isempty(cClq)
            Stop = true;
            break
        end

        % Adjust nodes of neighbouring cliques
        for iC = 1:numel(cClq)
            for iN = 1:obj.Clq(cClq(iC)).nF
                % Cov[C_i\cap{}C_j,v_p]
                Cov_CinCj_vp = cell(numel(cEl{iC}),1);
                for jC = 1:numel(cEl{iC})
                    lI = obj.Clq(cClq(iC)).nid==cEl{iC}(jC);
                    Cov_CinCj_vp{jC} =...
                            cell2mat(obj.Clq(cClq(iC)).CovF{lI,iN});
                end
                % Cov[D_k,v_p]
                Cov_Dk_vp = CFn{iC}*cell2mat(Cov_CinCj_vp);
                % E_{D_k}[v_{ik}]
                EDk_vk = cell2mat(obj.Clq(cClq(iC)).EF{iN}) +...
                                    Cov_Dk_vp'*(iVar_Dk*(Dk-E_Dk));
                % restore
                pob.Clq(cClq(iC)).EF{iN} = mat2cell(EDk_vk,...
                                obj.G.Nde(obj.Clq(cClq(iC)).nid(iN)).nV,1);
                for jN = 1:iN
                    % Cov[C_i\cap{}C_j,v_l]
                    Cov_CinCj_vq = cell(numel(cEl{iC}),1);
                    for jC = 1:numel(cEl{iC})
                        lI = obj.Clq(cClq(iC)).nid==cEl{iC}(jC);
                        Cov_CinCj_vq{jC} =...
                            cell2mat(obj.Clq(cClq(iC)).CovF{lI,jN});
                    end
                    % Cov[D_k,v_q]
                    Cov_Dk_vq = CFn{iC}*cell2mat(Cov_CinCj_vq);
                    % Cov_{D_k}[v_p,v_q]
                    CovDk_vp_vq =...
                        cell2mat(obj.Clq(cClq(iC)).CovF{iN,jN}) -...
                                Cov_Dk_vp'*(iVar_Dk*Cov_Dk_vq);
```

```
185                         % restore
186                         pob.Clq(cClq(iC)).CovF{iN,jN} = ...
187                                  mat2cell(CovDk_vp_vq,...
188                                   obj.G.Nde(obj.Clq(cClq(iC)).nid(iN)).nV,...
189                                   obj.G.Nde(obj.Clq(cClq(iC)).nid(jN)).nV);
190                         pob.Clq(cClq(iC)).CovF{jN,iN} = ...
191                                  mat2cell(CovDk_vp_vq',...
192                                   obj.G.Nde(obj.Clq(cClq(iC)).nid(jN)).nV,...
193                                   obj.G.Nde(obj.Clq(cClq(iC)).nid(iN)).nV);
194                     end
195                 end
196             end
197
198             % stop time
199             t = toc;
200             % output
201             for iC = 1:numel(cClq)
202                 fprintf('Clique %g updated in %.2f ...
                        seconds.\n',cClq(iC),t)
203             end
204         end
205
206         %% Restore partially-adjusted object
207         obj = pob;
208     end
209
210     return
```

The input 'obj' is an object of class 'JunctionTree', and the input structure 'F' identifies the nodes which were observed and stores the observed values. The code loops over the observed nodes, adjusting all moments stored on the graph using each node in turn, and using the moments after each adjustment as the prior for the next. For each observed node, the code first adjusts all members of a clique of which the observed node is a member. It then updates outwards from this clique: at each iteration of the inner 'while' loop, the code identifies all neighbours of the current clique set which have not yet been visited, and computes the covariance of the observed node with the nodes of these neighbouring cliques by identifying the intersection between the current clique set and its neighbours. The code stops and moves on to the next observed node when all cliques have been updated.

Bibliography

Robert J Adler. *Lecture Notes-Monograph Series*, volume 12. Institute of Mathematical Sciences, 1990. ISBN 094060017X.

James O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer, 1985. ISBN 9780387960982.

Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2003. ISBN 9780387310732. doi: 10.1073/pnas.0703993104.

S. J. Bourne and S. J. Oates. An activity rate model of induced seismicity within the Groningen Field (Part 1)- NAM report. Technical report, 2015. URL `https://nam-feitenencijfers.data-app.nl/download/rapport/8b6f2ff1-b98e-4148-a1db-bf06881579e5?open=true`.

S. J. Bourne, S. J. Oates, J. Bommer, B. Dost, J. van Elk, and D. Doornhof. A Monte Carlo Method for Probabilistic Hazard Assessment of Induced Seismicity due to Conventional Natural Gas Production. *Bulletin of the Seismological Society of America*, 105(3), 2015. doi: 10.1785/0120140302.

Jenny Brynjarsdottir and Anthony O'Hagan. Learning about physical parameters: The importance of model discrepancy. *Inverse Problems*, pages 1–21, 2014. ISSN 13616420. doi: 10.1088/0266-5611/30/11/114007.

John Rozier Cannon. *The One-Dimensional Heat Equation*. Cambridge University Press, 1985. ISBN 9780521302432.

Kathryn Chaloner and Isabella Verdinelli. Bayesian Experimental Design: A Re-

view. *Statistical Science*, 10(3):273–304, 1995. ISSN 0883-4237. doi: 10.1214/ss/1177009939.

Oksana A. Chkrebtii, David A Campbell, Ben Calderhead, and Mark A Girolami. Bayesian Solution Uncertainty Quantification for Differential Equations. *Bayesian Analysis*, 11(4):1239–1267, 2016. doi: 10.1214/16-BA1017. URL `http://arxiv.org/abs/1306.2365`.

Merlise A Clyde, Peter Muller, and Giovanni Parmigiani. Exploring Expected Utility Surfaces by Markov Chains. Technical report, 1996.

Stuart G. Coles. *An introduction to Statistical Modeling of Extreme Values*. 2001. ISBN 1852334592. doi: 10.1007/978-1-4471-3675-0.

Patrick R Conrad, Mark Girolami, Simo Särkkä, Andrew Stuart, and Konstantinos Zygalakis. Statistical analysis of differential equations : introducing probability measures on numerical solutions. *Statistics and Computing*, 27(4):1065–1082, 2017. ISSN 1573-1375. doi: 10.1007/s11222-016-9671-0. URL `https://link.springer.com/content/pdf/10.1007{%}2Fs11222-016-9671-0.pdf`.

Stefano Conti and Anthony O'Hagan. Bayesian emulation of complex multi-output and dynamic computer models. *Journal of Statistical Planning and Inference*, 140 (3):640–651, 2010. ISSN 03783758. doi: 10.1016/j.jspi.2009.08.006.

Peter S. Craig, Michael Goldstein, Jonathan C. Rougier, and Allan H. Seheult. Bayesian Forecasting for Complex Systems Using Computer Simulators. *Journal of the American Statistical Association*, 96(454):717–729, 2001. ISSN 0162-1459. doi: 10.1198/016214501753168370. URL `http://lysander.asa.catchword.org/vl=791500/cl=18/nw=1/rpsv/cw/asa/01621459/v96n454/s30/p717`.

Carl de Boor. B (asic)-spline basics. *Fundamental Developments of Computer-Aided Geometric Modeling*, pages 27–43, 1986. URL `ftp://ftp.cs.wisc.edu/Approx/bsplbasic.pdf`.

Bruno de Finetti. *Theory of probability: a critical introductory treatment*. Wiley, 1975. ISBN 0471926116. doi: 10.1038/163464a0.

Morris H. DeGroot. *Optimal Statistical Decisions*. Wiley, 1970. ISBN 9780471680291. doi: DOI:10.1002/0471729000.

Persi Diaconis. Bayesian Numerical Analysis- Stanford University, Report EF-SNSF261. Technical report, 1986. URL `https://statistics.stanford.edu/research/bayesian-numerical-analysis`.

Persi Diaconis and Donald Ylvisaker. Conjugate Priors for Exponentail families. *The Annals of Statistics*, 7(2):269–281, 1979.

Dove. Dove's Guide: Durham Cathedral, 2015. URL `http://dove.cccbr.org.uk/detail.php?searchString=Durham+Cath{&}Submit=+Go+{&}DoveID=DURHAM`.

R.R. Draxler. Determination of atmospheric diffusion parameters. *Atmospheric Environment (1967)*, 10(2):99–105, 1976. ISSN 00046981. doi: 10.1016/0004-6981(76)90226-2.

Robert Eymard, Thierry Gallouet, and Raphaele Herbin. Finite volume methods. *Handbook of Numerical Analysis*, 7:713–1018, 2000. ISSN 15708659. doi: 10.1016/S1570-8659(00)07005-8.

Adolf Fick. Ueber Diffusion. *Annalen der Physik*, 170(1):59–86, 1855. doi: 10.1002/andp.18551700105.

Thomas E. Fricker, Jeremy E. Oakley, and Nathan M. Urban. Multivariate Gaussian Process Emulators With Nonseparable Covariance Structures. *Technometrics*, 55: 47–56, 2013. ISSN 0040-1706. doi: 10.1080/00401706.2012.715835. URL `http://www.tandfonline.com/doi/abs/10.1080/00401706.2012.715835`.

Monserrat Fuentes, Peter Guttorp, and Peter Challenor. Statistical Assessment of Numerical Models. *International Statistical Review*, 71(2):201–221, 2003. doi: 10.1111/j.1751-5823.2003.tb00193.x.

G. A. Fuglstad, Finn Lindgren, Daniel Peter Simpson, and H. Rue. Exploring a New Class of Non-stationary Spatial Gaussian Random Fields with Varying Local Anisotropy. *Statistica Sinica*, 25(1):115–133, 2015. URL `http://arxiv.org/abs/1304.6949`.

Mark Girolami and Ben Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 73(2):123–214, 2011. ISSN 13697412. doi: 10.1111/j.1467-9868.2010.00765.x.

Michael Goldstein and Jonathan Rougier. Probabilistic formulations for transferring inferences from mathematical models to physical systems. *SIAM Journal on Scientific Computing*, 26(0):467–487, 2004. ISSN 1064-8275. doi: 10.1137/S106482750342670X. URL `http://dx.doi.org/10.1137/S106482750342670X`.

Michael Goldstein and Jonathan Rougier. Bayes Linear Calibrated Prediction for Complex Systems. *Journal Of The American Statistical Association*, 101(475):1132–1143, 2006. ISSN 0162-1459. doi: 10.1198/016214506000000203. URL `http://amstat.tandfonline.com/doi/full/10.1198/016214506000000203{%}5Cnpapers2://publication/uuid/EF7F6A79-EA3A-4A0A-8BFA-9275DFEDB349`.

Michael Goldstein and Jonathan Rougier. Reified Bayesian modelling and inference for physical systems. *Journal of Statistical Planning and Inference*, 139(3):1221–1239, 2009. ISSN 03783758. doi: 10.1016/j.jspi.2008.07.019.

Michael Goldstein and David Wooff. *Bayes Linear Statistics: Theory and Methods*. Wiley, Chichester, first edition, 2007. ISBN 978-0-470-01562-9. doi: 10.1002/9780470065662.

Thore Graepel. Solving Noisy Linear Operator Equations by Gaussian Processes: Application to Ordinary and Partial Differential Equations. *Twentieth International Conference on Machine Learning*, 2003.

Peter J Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732, 1995.

Peter Guttorp and Andrew Walden. On the evaluation of geophysical models. *Geophysical Journal of the Royal Astronomical Society*, 91(1):201–210, 1987. doi: 10.1111/j.1365-246X.1987.tb05220.x.

Ernst Hairer, Syvert P Norsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I*, volume 8. 1993. ISBN 978-3-540-56670-0. doi: 10.1007/978-3-540-78862-1. URL `http://www.springerlink.com/index/10.1007/978-3-540-78862-1`.

Philipp Hennig, Michael A Osborne, and Mark Girolami. Probabilistic Numerics and Uncertainty in Computations. *Proceedings of the Royal Society A*, 471(2179), 2015. doi: 10.1098/rspa.2015.0142.

Phillipp Henning and Christian J Schuler. Entropy Search for Information-Efficient Global Optimization. *Machine Learning Research*, 13(1999):1809–1837, 2012. ISSN 15324435.

Bill Hirst, Philip Jonathan, Fernando González del Cueto, David Randell, and Oliver Kosut. Locating and quantifying gas emission sources using remotely obtained concentration data. *Atmospheric Environment*, 74(October):141–158, 2013. ISSN 13522310. doi: 10.1016/j.atmosenv.2013.03.044.

Bill Hirst, David Randell, Matthew Jones, Philip Jonathan, Benjamin King, and Marcella Dean. A new technique for monitoring the atmosphere above onshore carbon storage projects that can estimate the locations and mass emission rates of detected sources. In *Energy Procedia*, volume 114, pages 3716–3728. The Author(s), 2017. doi: 10.1016/j.egypro.2017.03.1502. URL `http://dx.doi.org/10.1016/j.egypro.2017.03.1502`.

Xun Huan and Youssef M. Marzouk. Simulation-based optimal Bayesian experimental design for nonlinear systems. *Journal of Computational Physics*, 232(1):288–317, 2013. doi: 10.1016/j.jcp.2012.08.013. URL `http://arxiv.org/abs/1108.4146http://dx.doi.org/10.1016/j.jcp.2012.08.013`.

Xun Huan and Youssef M. Marzouk. Sequential Bayesian optimal experimental design via approximate dynamic programming. *Submitted*, 2016. URL `http://arxiv.org/abs/1604.08320`.

Nathan Huntley and Michael Goldstein. Assessing internal discrepancy for fast computer models. *To appear*, 2016.

Arieh Iserles. *A first course in the numerical analysis of differential equation*. 2008. ISBN 9780521734905. URL `http://books.google.com/books?hl=en{&}lr={&}id=7Zofw3SFTWIC{&}oi=fnd{&}pg=PR11{&}dq=A+first+course+in+the+numerical+analysis+of+differential+equation{&}ots=iLGGpB1TWb{&}sig=J803q-6DK5o{_}cSWHyPBpD0r4MoI`.

Richard Jeffrey. *Subjective Probability: the Real Thing*. Cambridge University Press, 2002. ISBN 9780521829717. doi: 10.1017/CBO9780511816161.

Philip Jonathan and Kevin Ewans. Statistical modelling of extreme ocean environments for marine design: A review. *Ocean Engineering*, 62(December 2012): 91–109, 2013. ISSN 00298018. doi: 10.1016/j.oceaneng.2013.01.004.

Matthew Jones, Michael Goldstein, Philip Jonathan, and David Randell. Bayes linear analysis for Bayesian optimal experimental design. *Journal of Statistical Planning and Inference*, 171:115–129, 2015. ISSN 03783758. doi: 10.1016/j.jspi.2015.10.011. URL `http://dx.doi.org/10.1016/j.jspi.2015.10.011`.

Matthew Jones, David Randell, Kevin Ewans, and Philip Jonathan. Statistics of extreme ocean environments : Non-stationary inference for directionality and other covariate effects. *Ocean Engineering*, 119:30–46, 2016. doi: 10.1016/j.oceaneng.2016.04.010.

Matthew Jones, Michael Goldstein, Philip Jonathan, and David Randell. Bayes Linear Analysis of Sequential Optimal Design Problems. *Forthcoming*, 2017.

M C Kennedy and A O'Hagan. Bayesian calibration of computer models. *J. R. Stat. Soc. Ser. B*, 63:425–464, 2001. doi: 10.1111/1467-9868.00294.

Tom W B Kibble and Frank H Berkshire. *Classical Mechanics*. Imperial College Press, London, 2004. ISBN 978-1-86094-435-2.

Matthew R.H. Killeya. *"Thinking inside the box": Using derivatives to improve Bayesian black box emulation of computer simulators with application to compartmental models*. PhD thesis, Durham University, 2004.

S.L. Lauritzen. *Graphical Models*. Clarendon Press, Oxford, 1996. ISBN 9780191591228.

S.L. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics*, 17(1):31–57, 1989.

Finn Lindgren, Håvard Rue, and Johan Lindström. An explicit link between gaussian fields and gaussian markov random fields: The stochastic partial differential equation approach. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 73(4):423–498, 2011. ISSN 13697412. doi: 10.1111/j.1467-9868.2011.00777.x.

J.L. Lund, A.R. Selby, and J.M. Wilson. The dynamics of bell towers - a survey in northeast England. *Transactions on the Built Environment*, 15, 1995. URL `https://www.witpress.com/Secure/elibrary/papers/STR95/STR95006FU2.pdf`.

J Mardia, J Kent, and J Bibby. *Multivariate Analysis*. 1979. ISBN 9780124712522.

J J Monaghan. Smoothed particle hydrodynamics. *Online*, 68:1703–1759, 2005. ISSN 0034-4885. doi: 10.1088/0034-4885/68/8/R01.

Peter Muller. Simulation Based Optimal Design. In J.M. Bernardo, J.O. Berger, A.P. Dawid, and A.F.M Smith, editors, *Bayesian Statistics*, pages 459–474. 1998. doi: 10.1016/S0169-7161(05)25017-4. URL `http://linkinghub.elsevier.com/retrieve/pii/S0169716105250174`.

Peter Müller, Don A. Berry, Andy P. Grieve, Michael Smith, and Michael Krams. Simulation-based sequential Bayesian design. *Journal of Statistical Planning and Inference*, 137(10):3140–3150, 2007. ISSN 03783758. doi: 10.1016/j.jspi.2006.05.021.

A. O'Hagan. BayesHermite quadrature. *Journal of Statistical Planning and Inference*, 29(3):245–260, 1991. ISSN 03783758. doi: 10.1016/0378-3758(91)90002-V. URL `http://www.sciencedirect.com/science/article/pii/037837589190002V`.

Anthony O'Hagan. Monte Carlo is Fundamentally Unsound. *Journal of the Royal Statistical Society. Series D*, 36(2):247–249, 1987.

D. B. Owen. A table of normal integrals. *Communications in Statistics- Simulation and Computation*, 9(4):389–419, 1980. doi: 10.1080/03610918008812164.

F. Pasquill. Atmospheric dispersion of pollution. *Quarterly Journal of the Royal Meteorological Society*, 97(414):369–395, 1971. ISSN 00359009. doi: 10.1002/qj.49709741402. URL http://doi.wiley.com/10.1002/qj.49709741402.

F Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):88–91, 1994. ISSN 00031305. doi: 10.1080/00031305.1994.10476030. URL http://www.jstor.org/stable/2684253{%}5Cnpapers2://publication/uuid/9ACCD11F-DDFC-4267-84CB-68C447DC1CCC.

H Raiffa and R Schlaifer. *Applied Statistical Decision Theory*. 1961. ISBN 047138349X.

D Randell, G Feld, K Ewans, and P Jonathan. Distributions of return values for ocean wave characteristics in the South China Sea using directionalseasonal extreme value analysis. *Environmetrics*, 26(6):442–450, 2015. doi: 10.1002/env.2350.

D Randell, K Turnbull, K Ewans, and P Jonathan. Bayesian inference for non-stationary marginal extremes. *Environmetrics*, 27(7):439–450, 2016. doi: 10.1002/env.2403.

Carl Edward Rasmussen and Zoubin Ghahramani. Bayesian Monte Carlo. *Advances in Neural Information Processing Systems 15*, (1):489–496, 2003. ISSN 10495258.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006. ISBN 0-262-18253-X. URL http://www.gaussianprocess.org/gpml/.

Christian P. Robert. The Metropolis-Hastings algorithm. (Mcmc):1–15, 2015. URL http://arxiv.org/abs/1504.01896.

Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer, New York, 1999. ISBN 9780387212395.

Stephan Rosswog. Astrophysical smooth particle hydrodynamics. *New Astronomy Reviews*, 53(4-6):78–104, 2009. ISSN 13876473. doi: 10.1016/j.newar.2009.08.007.

Jonathan Rougier. Efficient Emulators for Multivariate Deterministic Functions. *Journal of Computational and Graphical Statistics*, 17:827–843, 2008. doi: 10.1198/106186008X384032.

Havard Rue and Leonhard Held. *Gaussian Markov Random Fields: Theory and Applications*. 2005. ISBN 1584884320. doi: 10.1007/s00184-007-0162-3.

Thomas J Santner, Brian J Williams, and William I Notz. *The Design and Analysis of Computer Experiments*. Springer, New York, 1 edition, 2002. ISBN 1475737998, 9781475737998.

Leonard J Savage. *The Foundations of Statistics*. Dover, New York, 1972. ISBN 0-486-62349-1.

M. Schober, D. Duvenaud, and P. Hennig. Probabilistic ODE Solvers with Runge-Kutta Means. *Advances in Neural Information Processing Systems 27*, page 18, 2014. URL `http://arxiv.org/abs/1406.2582`.

Inanc Senocak, Nicolas W. Hengartner, Margaret B. Short, and W. Brent Daniel. Stochastic event reconstruction of atmospheric contaminant dispersion using Bayesian inference. *Atmospheric Environment*, 42:7718–7727, 2008. ISSN 13522310. doi: 10.1016/j.atmosenv.2008.05.024. URL `http://dx.doi.org/10.1016/j.atmosenv.2008.05.024`.

R. S. Smith, J. M. Gregory, and A. Osprey. A description of the FAMOUS (version XDBUA) climate model and control run. *Geoscientific Model Development*, 1(1): 53–68, 2008. ISSN 1991959X. doi: 10.5194/gmd-1-53-2008.

Richard Smith and Hugh Hunt. Vibration of bell towers excited by bell ringing a new approach to analysis. In *International Conference on Noise and Vibration*

Engineering, 2008. ISBN 9781615671915. URL http://www2.eng.cam.ac.uk/{~}hemh1/isma2008.pdf.

Ole R Sørensen, Henrik Kofoed-Hansen, Morten Rugbjerg, and Lars S Sørensen. A third-generation spectral wave model using an unstructured finite volume technique. *Proceedings of the 29th Intern. Conf. on Coastal Eng.*, pages 894–906, 2004. ISSN 01613782. doi: 10.1142/9789812701916-0071.

John M. Stockie. The Mathematics of Atmospheric Dispersion Modeling. *SIAM Review*, 53:349–372, 2011. ISSN 0036-1445. doi: 10.1137/10080991X. URL http://epubs.siam.org/doi/pdf/10.1137/10080991X{%}5Cnpapers2://publication/uuid/75D838F0-6E28-4CFA-965F-568BCC65CA79.

K van Thienen-Visser and J N Breunese. Induced seismicity of the Groningen gas field : History and recent developments. *The Leading Edge*, 34(6):664–671, 2015. doi: 10.1190/tle34060664.1.

AH Vermeulen, RH Bartels, and GR Heppler. Integrating products of B-splines. *SIAM Journal on Scientific Statistical Computing*, 13(4):1025–1038, 1992. URL http://epubs.siam.org/doi/pdf/10.1137/0913060.

Ian Vernon, Michael Goldstein, and Richard G. Bower. Galaxy formation: a Bayesian uncertainty analysis. *Bayesian Analysis*, 5(4):619–669, 2010. ISSN 1936-0975. doi: 10.1214/10-BA524. URL http://projecteuclid.org/euclid.ba/1340110846.

Peter Whittle. *Probability via Expectation*. Springer-Verlag, Cambridge, UK, 3 edition, 1992. ISBN 0-387-97764-3.

D. Williamson. *Policy making using computer simulators for complex physical systems ; Bayesian decision support for the development of adaptive strategies*. PhD thesis, Durham University, 2010.

D Williamson and M Goldstein. Bayesian policy support for adaptive strategies using computer models for complex physical systems. *Journal of the Operational*

Research Society, 63(8):1021–1033, 2012. ISSN 0160-5682. doi: 10.1057/jors.2011.110. URL http://dx.doi.org/10.1057/jors.2011.110.

Daniel Williamson, Michael Goldstein, Lesley Allison, Adam Blaker, Peter Challenor, Laura Jackson, and Kuniko Yamazaki. History matching for exploring and reducing climate model parameter space using observations and a large perturbed physics ensemble. *Climate Dynamics*, 41(7-8):1703–1729, 2013. ISSN 09307575. doi: 10.1007/s00382-013-1896-4.

Wolfram. Wolfram Mathematica: derivatives of the modified Bessel function of the second kind, 2017. URL http://functions.wolfram.com/Bessel-TypeFunctions/BesselK/20/ShowAll.html.

J. Woodhouse, J. C. Rene, C. S. Hall, L. T. W. Smith, F. H. King, and J. W. McClenahan. The Dynamics of a Ringing Church Bell. *Advances in Acoustics and Vibration*, 2012:1–19, 2012. ISSN 1687-6261. doi: 10.1155/2012/681787. URL http://www.hindawi.com/journals/aav/2012/681787/.

A. M. Yaglom. *Correlation Theory of Stationary and Related Random Functions*. Springer-Verlag, New york, 1986. ISBN 978-1-4612-9090-2.

Z-M Yin. New Methods for Simulation of Fractional Brownian Motion. *Journal of Computational Physics*, 127:66–72, 1996.

K Zickfeld, T Slawig, and S Rahmstorf. A low-order model for the response of the Atlantic thermohaline circulation to climate change. *Ocean Dynamics*, 54 (1):8–26, 2004. ISSN 1616-7341. doi: 10.1007/s10236-003-0054-7. URL http://link.springer.com/10.1007/s10236-003-0054-7.